A Guide to Corpus-Building for Applications

Early Release

Natural Language Annotation

for Machine Learning

James Pustejovsky
& Amber Stubbs

O'REILLY®

Natural Language Annotation for Machine Learning

James Pustejovsky and Amber Stubbs

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Natural Language Annotation for Machine Learning
by James Pustejovsky and Amber Stubbs

Revision History for the :

2012-03-06	Early release revision 1
2012-03-26	Early release revision 2

See *http://oreilly.com/catalog/errata.csp?isbn=9781449306663* for release details.

ISBN: 978-1-449-30666-3
1332788036

Table of Contents

Preface

This book is intended as a resource for people who are interested in using computers to help process natural language. A "natural language" refers to any language spoken by humans, either currently (e.g., English, Chinese, Spanish) or in the past (e.g., Latin, Greek, Sankrit). “Annotation” refers to the process of adding metadata information to the text in order to augment a computer’s abilities to perform Natural Language Processing (NLP). In particular, we examine how information can be added to natural language text through annotation in order to increase the performance of *machine learning* algorithms—computer programs designed to extrapolate rules from the information provided over texts in order to apply those rules to unannotated texts later on.

Natural Language Annotation for Machine Learning

More specifically, this book details the multi-stage process for building your own annotated natural language dataset (known as a corpus) in order to train machine learning (ML) algorithms for language-based data and knowledge discovery. The overall goal of this book is to show readers how to create their own corpus, starting with selecting an annotation task, creating the annotation specification, designing the guidelines, creating a "gold standard" corpus, and then beginning the actual data creation with the annotation process.

Because the annotation process is not linear, multiple iterations can be required for defining the tasks, annotations, and evaluations, in order to achieve the best results for a particular goal. The process can be summed up in terms of the MATTER *Annotation Development Process* cycle: Model, Annotate Train, Test, Evaluate, Revise. This books guides the reader through the cycle, and provides case studies for four different annotation tasks. These tasks are examined in detail to provide context for the reader and help provide a foundation for their own machine learning goals.

Additionally, this book provides lightweight, user-friendly software that can be used for annotating texts and adjudicating the annotations. While there are a variety of annotation tools available to the community, the Multi-purpose Annotation Environment (MAE), adopted in this book (and available to readers as a free download), was specifically designed to be easy to set up and get running, so readers will not be distracted from their goal with confusing documentation. MAE is paired with the Multi-document Adjudication Interface (MAI), a tool that allows for quick comparison of annotated documents.

Audience

This book is ideal for anyone interested in using computers to explore aspects of the information content conveyed by natural language. It is not necessary to have a programming or linguistics background to use this book, although a basic understanding of a scripting language like Python can make the MATTER cycle easier to follow. If you don't have any Python experience, we highly recommend the O'Reilly book *Natural Language Processing with Python* by Steven Bird, Ewan Klein, and Edward Loper, which provides an excellent introduction both to Python and to aspects of NLP that are not addressed in this book.

Organization of the Book

Chapter 1 of this book provides a brief overview of the history of annotation and machine learning, as well as short discussions of some of the different ways that annotation tasks have been used to investigate different layers of linguistic research. The rest of the book guides the reader through the MATTER cycle, from tips on creating a reasonable annotation goal in Chapter 2, all the way through evaluating the results of the annotation and machine learning stages and revising as needed. The last chapter gives a complete walkthrough of a single annotation project, and appendices at the back of the book provide lists of resources that readers will find useful for their own annotation tasks.

Software Requirements

While it's possible to work through this book without running any of the code examples provided, we do recommend having at least the Natural Language Toolkit (NLTK) installed for easy reference to some of the ML techniques discussed. The NLTK currently runs on Python versions from 2.4 to 2.7 (Python 3.0 is not supported at the time of this writing). For more information, see *www.nltk.org*.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This icon signifies a tip, suggestion, or general note.

This icon indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Book Title* by Some Author (O'Reilly). Copyright 2011 Some Copyright Holder, 978-0-596-xxxx-x."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

Safari Books Online is an on-demand digital library that lets you easily search over 7,500 technology and creative reference books and videos to find the answers you need quickly.

With a subscription, you can read any page and watch any video from our library online. Read books on your cell phone and mobile devices. Access new titles before they are available for print, and get exclusive access to manuscripts in development and post feedback for the authors. Copy and paste code samples, organize your favorites, download chapters, bookmark key sections, create notes, print out pages, and benefit from tons of other time-saving features.

O'Reilly Media has uploaded this book to the Safari Books Online service. To have full digital access to this book and others on similar topics from O'Reilly and other publishers, sign up for free at *http://my.safaribooksonline.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://www.oreilly.com/catalog/<catalog page>

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgements

From the Authors

We would like thank everyone at O'Reilly who helped us create this book, in particular Meghan Blanchette, Julie Steele, and Sarah Schneider for guiding us through the process of producing this book. We would also like to thank the students who participated in the Brandeis COSI 216 class from Spring 2011 for bearing with us as we worked through

the MATTER cycle with them: Karina Baeza Grossmann-Siegert, Elizabeth Baran, Bensiin Borukhov, Nicholas Botchan, Richard Brutti, Olga Cherenina, Russell Entrikin, Livnat Herzig, Sophie Kushkuley, Theodore Margolis, Alexandra Nunes, Lin Pan, Batia Snir, John Vogel, and Yaqin Yang.

CHAPTER 1

The Basics

It seems as though every day there are new and exciting problems that people have taught computers to solve, from Chess to Jeopardy to shortest-path driving directions. But there are still many tasks that computers cannot perform, particularly in the realm of understanding human language. Statistical methods have proven an effective way to approach these problems, but machine learning techiques often work better when the algorithms are provided with pointers to what is relevant about a dataset, rather than just massive amounts of data. When discussing natural language, these pointers often come in the form of annotations—metadata that provides additional information about the text. However, in order to teach a computer effectively, it's important to give it the right data, and for it to have enough data to learn from. The purpose of this book is to provide you with the tools to create good data for your own machine learning task. In this chapter we will cover:

- Why annotation is an important tool for linguists and computer scientists alike;
- How corpus linguistics became the field that it is today;
- The different areas of linguistics and how they relate to annotation and machine learning tasks;
- What a *corpus* is, and what makes a corpus balanced;
- How some classic machine learning problems are represnted with annotations;
- The basics of the Annotation Development Cycle.

The Importance of Language Annotation

Everyone knows that the Internet is an amazing resource for all sorts of information, that can teach you just about anything: juggling, programming, playing an instrument, and so on. However, there is another layer of information that the Internet contains, and that is how all those lessons (and blogs, forums, tweets, and so on) are being communicated. The web contains information in all forms of media—including texts, images, movies, sounds—and language is the communication medium that allows people to understand the content, and to link the content to other media. However, while computers are excellent at delivering this information to interested users, they are much less adept at understanding language itself.

Theoretical and computational linguistics are focused on unraveling the deeper nature of language and capturing the computational properties of linguistic structures. Human language technologies (HLT) attempt to adopt these insights and algorithms and turn them into functioning, high-performance programs that can impact the ways we interact with computers using language. With more and more people using the Internet every day, the amount of linguistic data available to researchers has increased significantly, allowing linguistic modeling problems to be viewed as machine learning tasks, rather than limited to the relatively small amounts of data that humans are capable of processing on their own.

However, it is not enough to simply provide a computer with a large amount of data and expect it to learn to speak—the data has to be prepared in such a way that the computer can more easily find patterns and inferences. This is usually done by adding relevant metadata to a dataset. Any metadata tag used to mark up elements of the dataset is called an *annotation* over the input. However, in order for the algorithms to learn efficiently and effectively, the annotation done on the data must be accurate, and relevant to the task the machine is being asked to perform. For this reason, the discipline of language annotation is a critical link in developing intelligent human language technologies.

Giving an algorithm too much information can slow it down and lead to inaccurate results. It's important to think carefully about what you are trying to accomplish, and what information is most relevant to that goal. Later in the book we will give examples of how to find that information.

Datasets of natural language are refered to as *corpora*, and a single set of data annotated the same way is called an *annotated corpus*. Annotated corpora can be used for training machine learning algorithms. In this chapter we will define what a corpus is, explain what is meant by an *annotation*, and describe the methodology used for enriching a linguistic data collection with annotations for machine learning.

The Layers of Linguistic Description

While it is not necessary to have formal linguistic training in order to create an annotated corpus, we will be drawing on examples of many different types of annotation tasks, and you will find this book more helpful if you have a basic understanding of the different aspects of language that are studied and used for annotations. *Grammar* is the name typically given to the mechanisms responsible for creating well-formed structures in language. Most linguists view grammar as itself consisting of distinct modules or systems, either by cognitive design or for descriptive convenience. These areas usually include: syntax, semantics, morphology, phonology (and phonetics), and the lexicon. Areas beyond grammar that relate to how language is embedded in human activity include: discourse, pragmatics, and text theory. These are described in more detail below:

Syntax
: The study of how words are combined to form sentences. This includes examining parts of speech and how they combine to make larger constructions.

Semantics
: The study of meaning in language. Semantics examines the relations between words and what they are being used to represent.

Morphology
: The study of units of meaning in a language. A “morpheme” is the smallest unit of language that has meaning or function, a definition that includes words, prefixes, affixes and other word structures that impart meaning.

Phonology
: The study of how phones are used in different languages to create meaning. Units of study include segments (individual speech sounds), features (the individual parts of segments), and syllables.

Phonetics
: The study of the sounds of human speech, and how they are made and perceived. “Phones” is the term for an individual sound, and a phone is essentially the smallest unit of human speech.

Lexicon
: Need definition here.

Discourse analysis
: The study of exchanges of information, usually in the form of conversations, particularly the flow of information across sentence boundaries.

Pragmatics
: The study of how context of text affects the meaning of an expression, and what information is necessary to infer a hidden or presupposed meaning.

Text structure analysis
: The study of how narratives and other textual styles are constructed to make larger textual compositions.

Throughout this book we will present examples of annotation projects that make use of various combinations of the different concepts outlined in the list above.

What is Natural Language Processing?

Natural Language Processing (NLP) is a field of computer science and engineering that has developed from the study of language and computational linguistics within the field of Artificial Intelligence. The goals of NLP are to design and build applications that facilitate human interaction with machines and other devices through the use of natural language. Some of the major areas of NLP include:

- Question Answering Systems (QAS): Imagine being able to actually ask your computer or your phone what time your favorite restaurant in New York stops serving dinner on Friday nights. Rather than typing in the (still) clumsy set of keywords into a search browser window, you could simply ask in plain, natural language—your *own*, whether it's English, Mandarin, or Spanish. (While systems like Siri for the iPhone are a good start to this process, it’s clear that Siri doesn’t fully understand all of natural language, just a subset of key phrases.)
- Summarization: This area includes applications that can take a collection of documents or emails, and produce a coherent summary of their content. Such programs also aim to provide snap elevator summaries of longer documents, and possibly even turn them into slide presentations.
- Machine Translation: The holy grail of NLP applications, this was the first major area of research and engineering in the field. Programs such as Google Translate are getting better and better, but the real killer app will be the BabelFish that translates in real-time when you're looking for the right train to catch in Beijing.
- Speech Recognition: This is one of the most difficult problems in NLP. There has been great progress in building models that can be used on your phone or computer to recognize spoken language utterances that are questions and commands. Unfortunately, while these Automatic Speech Recognition (ASR) systems are ubiquitous, they work best in narrowly-defined domains and don't allow the speaker to stray from the expected scripted input ("*Please say or type your card number now.* ")
- Document Classification: This is one of the most successful areas of NLP, wherein the task is to identify which category (or *bin*) a document should be put in. This has proved enormously useful for applications such as spam filtering, news article classification, and movie reviews, among others. One reason this has had such a big impact is the relative simplicity of the learning models needed for training the algorithms that do the classification.

As we mentioned in the Preface, the Natural Language Toolkit (NLTK), described in the O'Reilly book *Natural Language Processing with Python*, is a wonderful introduction to the essential techniques necessary to build many of the applications listed above. One of the goals of this book is to give you the knowledge to build specialized language corpora (i.e., training and test datasets) that are necessary for developing such applications.

A Brief History of Corpus Linguistics

In the mid-twentieth century, linguistics was practiced primarily as a descriptive field, used to study structural properties within a language and typological variations between languages. This work resulted in fairly sophisticated models of the different informational components comprising linguistic utterances. As in the other social sciences, the collection and analysis of data was also being subjected to quantitative techniques from statistics. In the 1940s, linguists such as Bloomfield were starting to think that language could be explained in probabilistic and behaviorist terms. Empirical and statistical methods became popular in the 1950s, and Shannon's information-theoretic view to language analysis appeared to provide a solid quantitative approach for modeling qualitative descriptions of linguistic structure.

Unfortunately, the development of statistical and quantitative methods for linguistic analysis hit a brick wall in the 1950s. This was due primarily to two factors. First, there was the problem of data availability. One of the problems with applying statistical methods to the language data at the time was that the datasets were generally so small that it was not possible to make interesting statistical generalizations over large numbers of linguistic phenomena. Secondly, and perhaps more importantly, there was a general shift in the social sciences from data-oriented descriptions of human behavior to introspective modeling of cognitive functions.

As part of this new attitude towards human activity, the linguist Noam Chomsky focused on both a formal methodology and a theory of linguistics that not only ignored quantitative language data, but also claimed that it was misleading for formulating models of language behavior (Chomsky, 1957).

Timeline of Corpus Linguistics

- **1950's**: Descriptive linguists compile collections of spoken and written utterances of various languages from field research. Literary researchers begin compiling systematic collections of the complete works of different authors. Key Word in Context (KWIC) is invented as a means of indexing documents and creating concordances.
- **1960's**: Kucera and Francis publish *A Standard Corpus of Present-Day American English* (the *Brown Corpus*), the first broadly available large corpus of language texts. Work in Information Retrieval (IR) develops techniques for statistical similarity of document content.
- **1970's**: Stochastic models developed from speech corpora make Speech Recognition Systems possible. The vector space model is developed for document indexing. The London-Lund Corpus (LLC) is developed through the work of the *Survey of English Usage*.
- **1980's**: The Lancaster-Oslo-Bergen Corpus (LOB) is compiled. Designed to match the Brown corpus in terms of size and genres. The COBUILD (Collins Bir-

mingham University International Language Database) dictionary is published, the first based on examining usage from a large English corpus, the Bank of English. The Survey of English Usage Corpus inspires the creation of a comprehensive corpus-based grammar, *Grammar of English*. The Child Language Data Exchange System (CHILDES) corpus is released as a repository for first language acquisition data.

- **1990's**: The Penn TreeBank is released. This is a corpus of tagged and parsed sentences of naturally occuring English (4.5 M words). The British National Corpus (BNC) is compiled and released as the largest corpus of English to date (100M words). The Text Encoding Initiative (TEI) is established to develop and maintain a standard for the representation of texts in digital form.
- **2000's**: As the World Wide Web grows, more data is available for statistical models for Machine Translation and other applications. The American National Corpus project releases a 22M subcorpus, and the Corpus of Contemporary American English is released (400M words). Google releases their Google n-gram corpus of 1 trillion word tokens from public webpages, which has n-grams (up to 5) along with their frequencies.
- **2010's**: International standards organizations, such as the ISO, begin recognizing and co-developing text encoding formats that are being used for corpus annotation efforts. The web continues to make enough data available to build models for a whole new range of linguistic phenomena. Entirely new forms of text corpora become available as a resource, such as Twitter, Facebook, and blogs.

This view was very influential throughout the 1960s and 1970s, largely because the formal approach was able to develop extremely sophisticated rule-based language models using mostly introspective (or self-generated) data. This was a very attractive alternative to trying to create statistical language models on the basis of still relatively small data sets of linguistic utterances from the existing corpora in the field. Formal modeling and rule-based generalizations, in fact, have always been an integral step in theory formation, and in this respect Chomsky's approach on how to do linguistics has yielded rich and elaborate models of language.

Theory construction, however, also involves testing and evaluating your hypotheses against observed phenomena. As more linguistic data has gradually become available, something significant has changed in the way linguists look at data. The phenomena are now observable in millions of texts and billions of sentences over the web, and this has left little doubt that quantitative techniques can be meaningfully applied to both test and create the language models correlated with the datasets. This has given rise to the modern age of corpus linguistics. As a result, the corpus is the entry point from which all linguistic analysis will be done in the future.

You gotta have data! As the philosopher of science, Thomas Kuhn said: "When measurement departs from theory, it is likely to yield mere numbers, and their very neutrality makes them particularly sterile as a source of remedial suggestions. But numbers register the departure from theory with an authority and finesse that no qualitative technique can duplicate, and that departure is often enough to start a search." (Kuhn, 1961)

The assembly and collection of texts into more coherent datasets that we can call *corpora* started in the 1960s.

Some of the most important corpora are listed in Table 1-1.

Name of Corpus	Year Published	Size	Collection Contains
British National Corpus (BNC)	1991-1994	100M words	cross-section of British English, spoken and written
Corpus of Contemporary American English (COCA)	2008	425M words	spoken, fiction, popular magazines, academic texts
American National Corpus (ANC)	2003	22M words	spoken and written texts

What is a Corpus?

A corpus is a collection of machine-readable texts that have been produced in a natural communicative setting. They have been sampled to be *representative* and *balanced* with respect to particular factors, for example by genre–newspaper articles, literary fiction, spoken speech, blogs and diaries, legal documents. A corpus is said to be "representative of a language variety" if the content of the corpus can be generalized to that variety (Leech, 1991).

This is not as circular as it may sound. Basically, if the content of the corpus, defined by specifications of linguistic phenomena examined or studied, reflects that of the larger population from which it is taken, then we can say that it "represents that language variety."

The notion of a corpus being *balanced* is an idea that has been around since the 1980s, but it is still rather a fuzzy notion and difficult to define strictly. Atkins and Ostler (1992) propose a formulation of attributes that can be used to define the types of text, and thereby contribute to creating a balanced corpus.

Two well-known corpora can be compared for their effort to balance the content of the texts. The Penn Treebank (Marcus et al, 1993) is a one-million-word corpus that contains texts from four sources: the Wall Street Journal, the Brown Corpus, ATIS, and the Switchboard Corpora. By contrast, the British National Corpus (BNC) is a 100-million-word corpus that contains texts from a broad range of genres, domains, and media.

The most diverse subcorpus within the Penn TreeBank is the Brown Corpus, which is a one-million-word corpus consisting of 500 English text samples, each one approximately 2,000 words. It was collected and compiled by Henry Kucera and W. Nelson Francis from Brown University (hence its name) from a broad range of contemporary American English in 1961. In 1967, they released a fairly extensive statistical analysis of the word frequencies and behavior within the corpus, the first of its kind in print, as well as the Brown Corpus Manual (Francis and Kucera, 1964).

There has never been any doubt that all linguistic analysis must be grounded on specific datasets. What has recently emerged is the realization that all linguistics will be bound to corpus-oriented techniques, one way or the other. Corpora are becoming the standard data exchange for discussing linguistic observations and theoretical generalizations, and certainly for evaluation of systems, both statistical and rule-based.

Below is a table that shows how the Brown Corpus compares to other corpora that are also still in use.

Brown Corpus	500 English text samples; 1 million words	part-of-speech tagged data; 80 different tags used
Child Language Data Exchange System (CHILDES)	20 language represented; thousands of texts	phonetic transcriptions of conversations with children from around the world
Lancaster-Oslo-Bergen Corpus	500 British English text samples; around 2,000 words each	part-of-speech tagged data; a British version of the Brown corpus

Looking at the way the files of the Brown Corpus can be categorized gives us an idea of what sorts of data were used to represent the English language. The top two general data categories are Informative and Imaginitive, as see in Table Table 1-1.

Table 1-1. Brown Corpus: general categories

Informative Prose	374 samples
Imaginative Prose	126 samples

These two domains are further distinguished into the following topic areas:

- **Informative**: Press: reportage (44), Press: editorial (27), Press: reviews (17), Religion (17), Skills and Hobbies (36), Popular Lore (48), Belles Lettres, Biography, Memoirs (75), Miscellaneous (30), Natural Sciences (12), Medicine (5), Mathematics (4), Social and Behavioral Sciences (14), Political Science, Law, Education (15), Humanities (18), Technology and Engineering (12).

- **Imaginative**: General Fiction (29), Mystery and Detective Fiction (24), Science Fiction (6), Adventure and Western Fiction (29), Romance and Love Story (29) Humor (9).

Similarly, the British National Corpus (BNC) can be categorized into *informative* and *imaginitive* prose, and further into subdomains such as *educational*, *public*, *business*, etc. A further discussion of how the BNC can be categorized can be found in "Distributions within corpora" on page 45.

As you can see from the numbers given for the Brown Corpus, not every category is equally represented, which seems to be a violation of the rule of "representative and balanced" that we discussed before. However, these corpora were not assembled with a specific task in mind; rather, they were meant to represent written and spoken language as a whole. Because of this, they attempt to embody a large cross-section of existing texts, though whether they succeed in representing percentages of texts in the world is debateable (but also not terribly important).

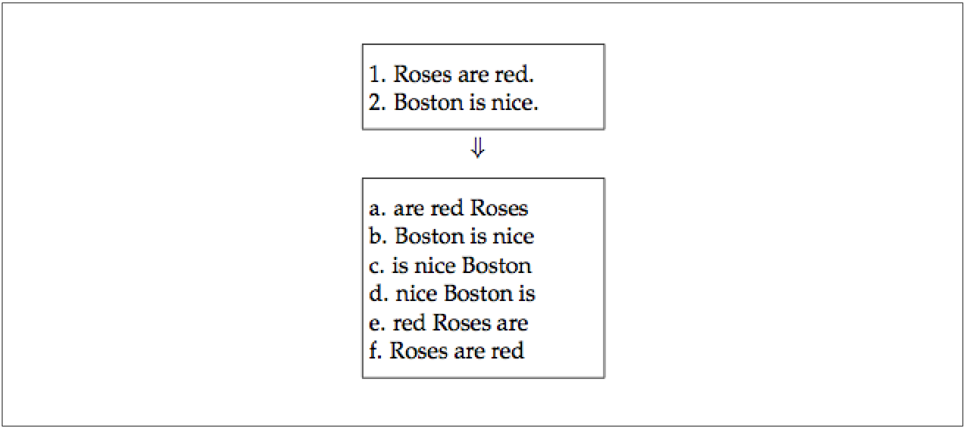
For your own corpus, you may find yourself wanting to cover a wide variety of text, but it is likely that you will have a more specific task domain, and so your potential corpus will not need to include the full range of human expression. The switchboard corpus is an example of a corpus that was collected for a very specific purpose—speech recognition for phone operation—and so was balanced and representative of the different sexes and all different dialects in the United States.

Early Use of Corpora

One of the most common uses of corpora from the early days was the construction of *concordances*. These are alphabetical listings of the words in an article or text collection with references given to the passage in which they occur. Concordances position a word within its context, and thereby make it much easier to study how it is used in a language, both syntactically and semantically. In the 1950s and 1960s, programs were written to automatically create concordances for the contents of a collection, and the result of these automatically-created indexes were called "Keyword in Context" Indexes, or *KWIC Indexes*. A KWIC Index is an index created by sorting the words in an article or a larger collection such as a corpus, and aligning the words in a format so that they can be searched alphabetically in the index. This was a relatively efficient means for searching a collection before full-text document search became available.

The way a KWIC index works is as follows. The input to a KWIC system is a file or collection structured as a sequence of lines. The output is a sequence of lines, circularly shifted and presented in alphabetical order of the first word. For an example, consider a short article of two sentences, shown in Figure 1-1 with the KWIC index output that is generated.

Another benefit of concordancing is that, by displaying the keyword in the surrounding context, you can visually inspect how a word is being used in a given sentence. To take

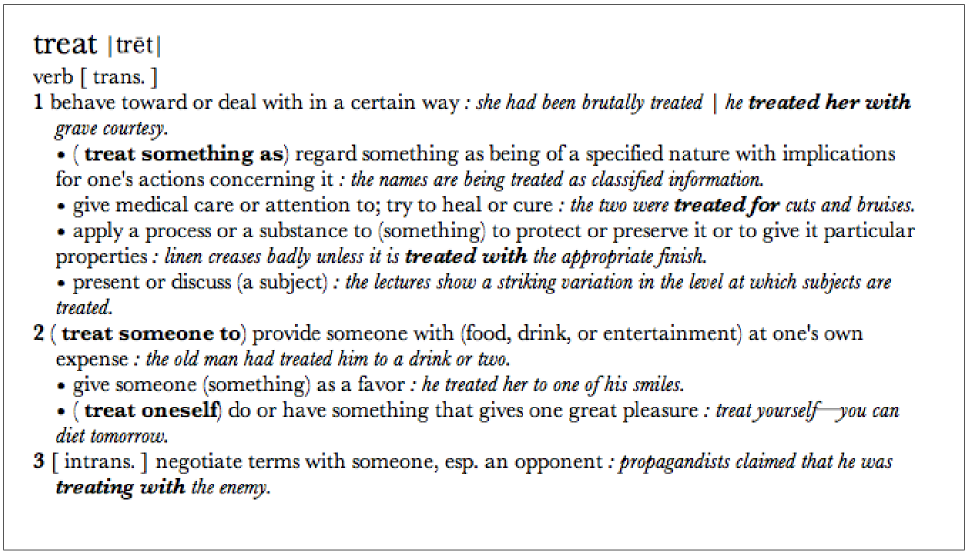


1. Roses are red.
2. Boston is nice.

⇓

a. are red Roses
b. Boston is nice
c. is nice Boston
d. nice Boston is
e. red Roses are
f. Roses are red

Figure 1-1. Example of a KWIC index

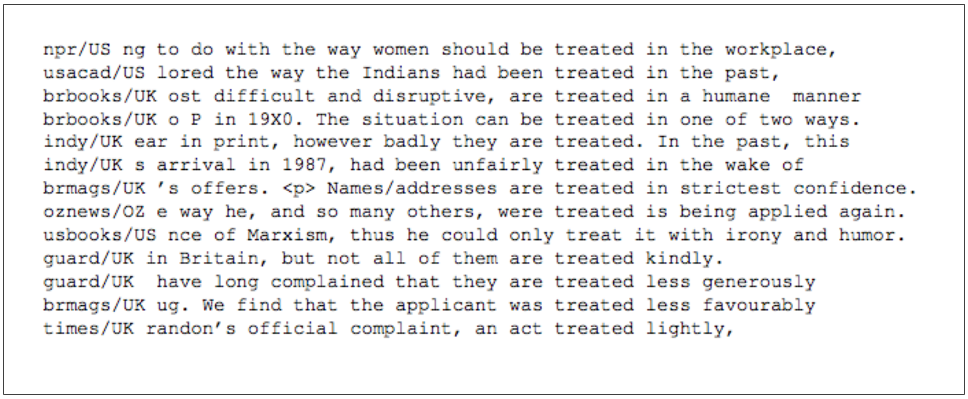


treat |trēt|
verb [trans.]
1 behave toward or deal with in a certain way : *she had been brutally treated* | *he* ***treated her with*** *grave courtesy.*
• (**treat something as**) regard something as being of a specified nature with implications for one's actions concerning it : *the names are being treated as classified information.*
• give medical care or attention to; try to heal or cure : *the two were* ***treated for*** *cuts and bruises.*
• apply a process or a substance to (something) to protect or preserve it or to give it particular properties : *linen creases badly unless it is* ***treated with*** *the appropriate finish.*
• present or discuss (a subject) : *the lectures show a striking variation in the level at which subjects are treated.*
2 (**treat someone to**) provide someone with (food, drink, or entertainment) at one's own expense : *the old man had treated him to a drink or two.*
• give someone (something) as a favor : *he treated her to one of his smiles.*
• (**treat oneself**) do or have something that gives one great pleasure : *treat yourself—you can diet tomorrow.*
3 [intrans.] negotiate terms with someone, esp. an opponent : *propagandists claimed that he was* ***treating with*** *the enemy.*

Figure 1-2. Senses of the word "treat"

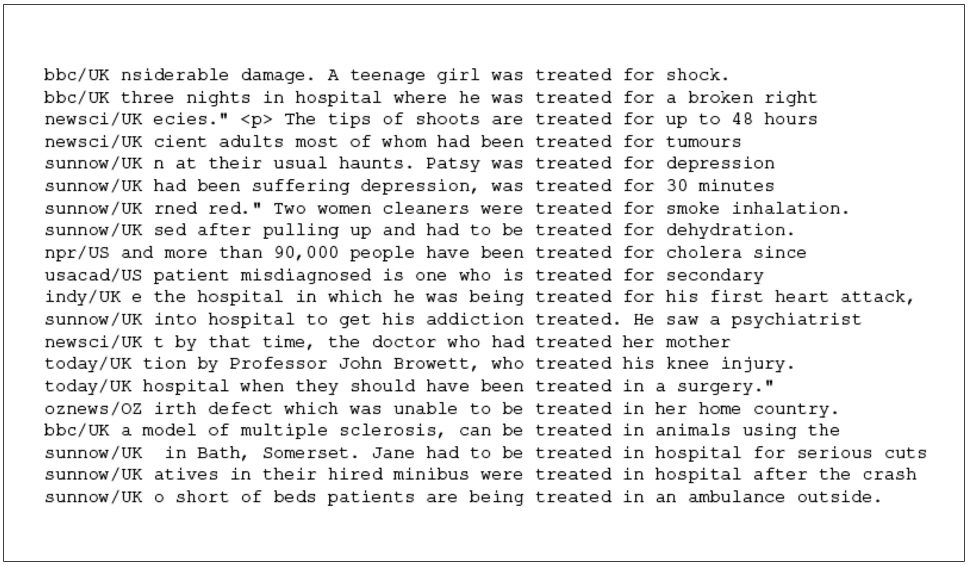
a specific example, consider the different meanings of the English verb *treat*. Specifically, let's look at the first two senses within sense (1) from the dictionary entry shown in Figure 1-2.

Now's let's look at the concordances compiled or this verb from the British National Corpus (BNC), as differentiated by these two senses.



```
npr/US ng to do with the way women should be treated in the workplace,
usacad/US lored the way the Indians had been treated in the past,
brbooks/UK ost difficult and disruptive, are treated in a humane  manner
brbooks/UK o P in 19X0. The situation can be treated in one of two ways.
indy/UK ear in print, however badly they are treated. In the past, this
indy/UK s arrival in 1987, had been unfairly treated in the wake of
brmags/UK 's offers. <p> Names/addresses are treated in strictest confidence.
oznews/OZ e way he, and so many others, were treated is being applied again.
usbooks/US nce of Marxism, thus he could only treat it with irony and humor.
guard/UK in Britain, but not all of them are treated kindly.
guard/UK  have long complained that they are treated less generously
brmags/UK ug. We find that the applicant was treated less favourably
times/UK randon's official complaint, an act treated lightly,
```

Figure 1-3. Sense 1a for the verb ***treat***



```
bbc/UK nsiderable damage. A teenage girl was treated for shock.
bbc/UK three nights in hospital where he was treated for a broken right
newsci/UK ecies." <p> The tips of shoots are treated for up to 48 hours
newsci/UK cient adults most of whom had been treated for tumours
sunnow/UK n at their usual haunts. Patsy was treated for depression
sunnow/UK had been suffering depression, was treated for 30 minutes
sunnow/UK rned red." Two women cleaners were treated for smoke inhalation.
sunnow/UK sed after pulling up and had to be treated for dehydration.
npr/US and more than 90,000 people have been treated for cholera since
usacad/US patient misdiagnosed is one who is treated for secondary
indy/UK e the hospital in which he was being treated for his first heart attack,
sunnow/UK into hospital to get his addiction treated. He saw a psychiatrist
newsci/UK t by that time, the doctor who had treated her mother
today/UK tion by Professor John Browett, who treated his knee injury.
today/UK hospital when they should have been treated in a surgery."
oznews/OZ irth defect which was unable to be treated in her home country.
bbc/UK a model of multiple sclerosis, can be treated in animals using the
sunnow/UK  in Bath, Somerset. Jane had to be treated in hospital for serious cuts
sunnow/UK atives in their hired minibus were treated in hospital after the crash
sunnow/UK o short of beds patients are being treated in an ambulance outside.
```

Figure 1-4. Sense 1b for the verb ***treat***

These concordances were compiled using the *WordSketch Engine*, by the lexicographer Patrick Hanks, and are part of a large resource of sentence patterns using a technique called *Corpus Pattern Analysis*. (Pustejovsky et al., 2004; Hanks and Pustejovsky, 2005).

What is striking when one examines the concordance entries for each of these senses is how distinct the contexts of use are. These are presented below.

Your Turn: NLTK provides functionality for creating concordances. The easiest way to make a concordance is to simply load the pre-processed texts in NLTK, then use the concorance function, like this:

```
>>> from nltk.book import *
>>> text6.concordance("Ni")
```

If you have your own set of data for which you would like to create a concordance, then the process is a little more involved: you will need to read in your files and use the NLTK functions to process them before you can create your own concordance. Sample code for a corpus of text files is provided below:

```
>>> corpus_loc = '/home/me/corpus/'
>>> docs = nltk.corpus.PlaintextCorpusReader(corpus_loc,'.*\.txt')
```

You can see if the files were read by checking what fileids are present:

```
>>> print docs.fileids()
```

Next, process the words in the files and then use the concordance function to examine the data

```
>>> docs_processed = nltk.Text(docs.words())
>>> docs_processed.concordance("treat")
```

Corpora Today

When did researchers start actually using corpora for modeling language phenomena and training algorithms? Starting in the 1980s, researchers in speech recognition began compiling enough spoken language data to create language models (from transcriptions using n-grams and Hidden Markov Models) that worked well enough to recognize a limited vocabulary of words in a very narrow domain. In the 1990s, work in machine translation began to see the influence of larger and larger datasets, and with this the rise of statistical language modeling for translation.

Eventually, both memory and computer hardware became sophisticated enough to collect and analyze larger and larger datasets of language fragments. This entailed being able to create statistical language models that actually perform with some reasonable accuracy for different natural language tasks.

As one example of the increasing availability of data, Google has recently released the *Google Ngram Corpus*. The Google Ngram dataset allows users to search for single words (unigrams) or collocations of up to five words (5-grams). The dataset is available for download from the Linguistic Data Consortium, and directly from Google. It is also viewable online through the Google NGram Viewer. The Ngram dataset consists over over 1 trillion tokens (words, numbers, etc) taken from publically available websites and sorted by year, making it easy to view trends in language use. In addition to English, Google provides ngrams for Chinese, French, German, Hebrew, Russian, and Spanish, as well as subsets of the English corpus such as American English and English Fiction.

N-grams are sets of items (often words, but can be letters, phonemes, etc) that are part of a sequence. By examining how often the items occur together we can learn about their usage in a language, and make predictions about what would likely follow a given sequence (using ngrams for this purpose is called *n-gram modeling*).

N-grams are applied in a variety of ways every day, such as websites that provide search suggestions once a few letters are typed in and determining likely substitutions for spelling errors. They are also used in speech disambiguation—if a person speaks unclearly but utters a sequence that does not commonly (or ever) occur in the language being spoken, an n-gram model can help recognize that problem and find words that were probably what the speaker intended to say.

Kinds of Annotation

Consider first the different components of syntax that can be annotated. These include *part of speech (POS)*, *phrase structure*, and *dependency structure*. Examples of each of these are shown below. There are many different tagsets for the parts of speech of a language that you can choose from.

Table 1-2. Number of POS tags in different corpora

Tagset	Size	Date
Brown	77	1964
Penn	36	1992
LOB	132	1980s
London-Lund Corpus	197	1982

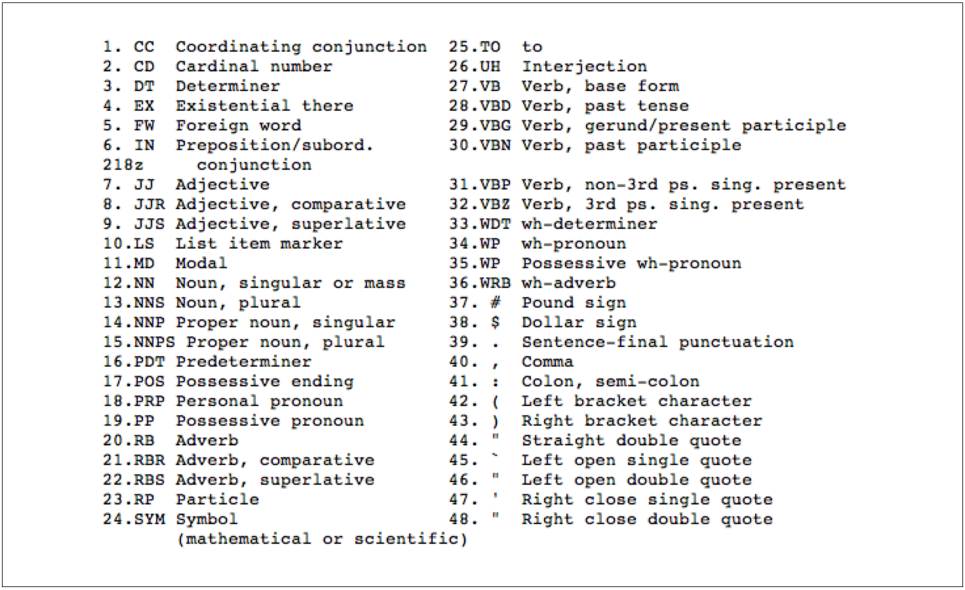
The tagset illustrated below is taken from the Penn Treebank, and is the basis for all subsequent annotation over that corpus.

The process of part-of-speech (POS) tagging is assigning the right lexical class marker(s) to all the words in a sentence (or corpus). This is illustrated in a simple example, "The waiter cleared the plates from the table."

POS tagging is a critical step in many NLP applications since it is important to know what category a word is assigned to in order to perform subsequent analysis on it, such as:

speech synthesis
: is the word a noun or a verb? “object”, “overflow”, “insult”, “suspect”, etc. Without context, each of these words could be either a noun or a verb.

parsing
: you need POS tags in order to make larger syntactic units: is "clean dishes" a noun phrase or an imperative verb phrase?



1. CC	Coordinating conjunction	25.TO	to
2. CD	Cardinal number	26.UH	Interjection
3. DT	Determiner	27.VB	Verb, base form
4. EX	Existential there	28.VBD	Verb, past tense
5. FW	Foreign word	29.VBG	Verb, gerund/present participle
6. IN	Preposition/subord.	30.VBN	Verb, past participle
218z	conjunction		
7. JJ	Adjective	31.VBP	Verb, non-3rd ps. sing. present
8. JJR	Adjective, comparative	32.VBZ	Verb, 3rd ps. sing. present
9. JJS	Adjective, superlative	33.WDT	wh-determiner
10.LS	List item marker	34.WP	wh-pronoun
11.MD	Modal	35.WP	Possessive wh-pronoun
12.NN	Noun, singular or mass	36.WRB	wh-adverb
13.NNS	Noun, plural	37. #	Pound sign
14.NNP	Proper noun, singular	38. $	Dollar sign
15.NNPS	Proper noun, plural	39. .	Sentence-final punctuation
16.PDT	Predeterminer	40. ,	Comma
17.POS	Possessive ending	41. :	Colon, semi-colon
18.PRP	Personal pronoun	42. (	Left bracket character
19.PP	Possessive pronoun	43.)	Right bracket character
20.RB	Adverb	44. "	Straight double quote
21.RBR	Adverb, comparative	45. `	Left open single quote
22.RBS	Adverb, superlative	46. "	Left open double quote
23.RP	Particle	47. '	Right close single quote
24.SYM	Symbol	48. "	Right close double quote
	(mathematical or scientific)		

Figure 1-5. The Penn Treebank Tagset

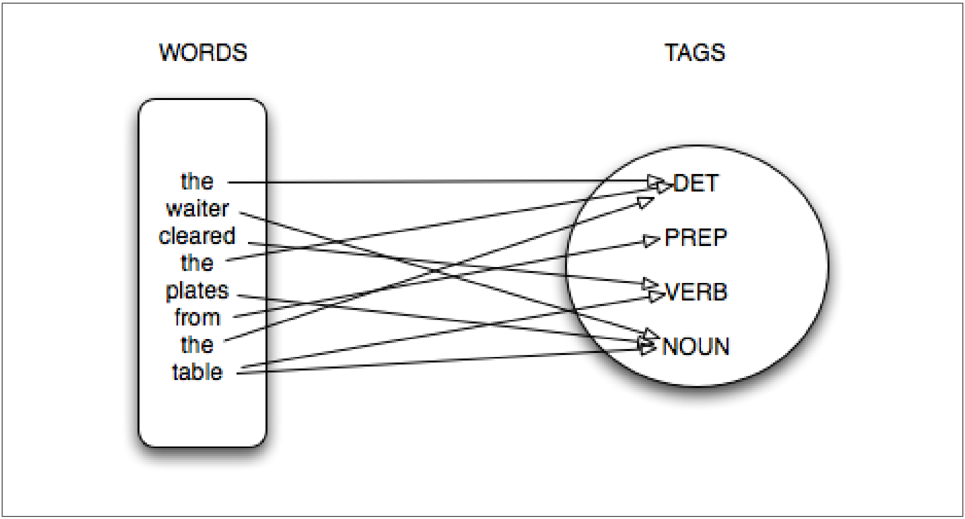


Figure 1-6. Examples of part-of-speech tagging

"Clean dishes are on in the cabinet." versus a note on the fridge: *Clean dishes before going to work!*

machine translation
: getting the POS tags and the subsequent parse right makes all the difference when translating the expressions above into another language, e.g., French: "des assiettes propres" (clean dishes) vs. "Fais la vaisselle!" (clean the dishes!).

Consider how these tags are used in a sentence from the corpus known as the Penn Treebank (Marcus et al, 1993 on page 86), illustrated below.

"From the beginning, it took a man with extraordinary qualities to succeed in Mexico," says Kimihide Takimura, president of Mitsui group's Kensetsu Engineering Inc. unit.

"/" From/IN the/DT beginning/NN ,/, it/PRP took/VBD a/DT man/NN with/IN extraordinary/JJ qualities/NNS to/TO succeed/VB in/IN Mexico/NNP ,/, "/" says/VBZ Kimihide/NNP Takimura/NNP ,/, president/NN of/IN Mitsui/NNS group/NN 's/POS Kensetsu/NNP Engineering/NNP Inc./NNP unit/NN ./.

Identifying the correct parts of speech in a sentence is a necessary step in building many natural language applications, such as parsers, Named Entity Recognizers, Question-Answering systems, and Machine Translation systems. It is also an important step towards identifying larger structural units such as phrase structure.

Your Turn: Use the NTLK tagger to assign part-of-speech tags to the example sentence shown here, and then with other sentences that might be more ambiguous.

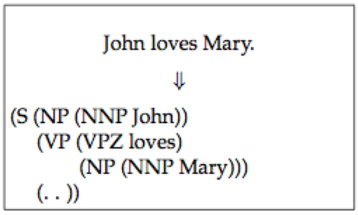
```
>>> from nltk import pos_tag, word_tokenize
>>> pos_tag(word_tokenize("This is a test."))
```

Look for places where the tagger doesn't work, and think about what rules might be causing these errors. For example, what happens when you try "Clean dishes are on in the cabinet." and "Clean dishes before going to work!"

While words have labels associated with them (the Part-of-Speech tags mentioned above), specific sequences of words also have labels that be associated with them. This is called *syntactic bracketing* (or labeling) and is the structure that organizes all the words we hear into coherent phrases. As mentioned above, syntax is the name given to the structure associated with a sentence. The Penn Treebank is an annotated corpus with syntactic bracketing explicitly marked over the text. An example annotation is shown in the box below.

This is a bracketed representation of the syntactic tree structure, which is shown below.

Notice that a syntactic bracketing introduces two relations between the words in a sentence: order (precedence), and hierarchy (dominance). For example, the tree structure above encodes these relations by the very nature of a tree as a directed acyclic graph


John loves Mary.

⇓

```
(S (NP (NNP John))
   (VP (VPZ loves)
       (NP (NNP Mary)))
   (. . ))
```

Figure 1-7. Syntactic bracketing

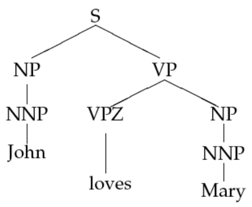


Figure 1-8. Syntactic Tree Structure

(DAG). In a very compact form, the tree above captures the precedence and dominance relations given in the list below;

> {Dom(NNP1,John), Dom(VPZ,loves), Dom(NNP2,Mary), Dom(NP1,NNP1), Dom(NP2,NNP2), Dom(S,NP1), Dom(VP,VPZ), Dom(VP,NP2), Dom(S,VP),
>
> Prec(NP1,VP), Prec(VPZ,NP2)}

Any sophisticated natural language application requires some level of syntactic analysis, including machine translation. If the resources for *full parsing* (such as that shown above) are not available, then some sort of *shallow parsing* can be used. This is when partial syntactic bracketing is applied to sequences of words, without worrying about the details of the structure inside of a phrase. We will return to this idea in later chapters.

In addition to part of speech tagging and syntactic bracketing, it is useful to annotate texts in a corpus for their semantic value, i.e., what the words mean in the sentence. We can distinguish two kinds of annotation for semantic content within a sentence: what something *is*, and what *role* something plays. These are described in more detail below.

Semantic Typing
: a word or phrase in the sentence is labeled with a type identifier (from a reserved vocabulary or ontology), indicating what it denotes.

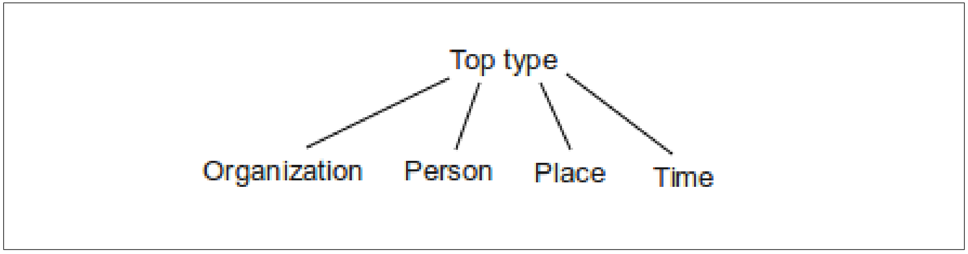


Figure 1-9. A simple ontology

Semantic Role Labeling
: a word or phrase in the sentence is identified as playing a specific semantic role relative to a role assigner, such as a verb.

Let's consider what annotation using these two strategies would look like, starting with semantic types. Types are commonly defined using an ontology, such as that shown in Figure 1-9.

> The word **ontology** has its roots in philosophy, but ontologies also have a place in computational linguistics, where they are used to create categorized heirarchies that group similar concepts and objects. By assigning words semantic types in an ontology, we can create relationships between different branches of the ontology, and determine whether linguistic rules hold true when applied to all the words in a category.

This ontology is rather simple, with a small set of categories. However, even this small ontology can be used to illustrate some interesting features of language. Consider the following example, with semantic types marked:

> [Ms. Ramirez]$_{\text{Person}}$ of [QBC Productions]$_{\text{Organization}}$ visited [Boston]$_{\text{Place}}$ on [Saturday]$_{\text{Time}}$, where she had lunch with [Mr. Harris]$_{\text{Person}}$ of [STU Enterprises]$_{\text{Organization}}$ at [1:15 pm]$_{\text{Time}}$.

From this small example, we can start to make observations about how these objects interact with each other. People can visit places, people have "of" relationships with organizations, and lunch can happen on Saturday at 1pm. Given a large enough corpus of similarly labeled sentences, we can start to detect patterns in usage that will tell us more about how these labels do and do not interact.

A corpus of these examples can also tell us where, perhaps, our categories need to be expanded. There are two "times" in this sentence: Saturday and 1pm. We can see that events can occur "on" Saturday, but "at" 1pm. A larger corpus would show that this pattern remains true with other days of the week and hour designations—there is a difference in usage here that cannot be inferred from the semantic types. However, not all ontologies will capture all information—the applications of the ontology will determine whether it is important to capture the difference between Saturday and 1pm.

The annotation strategy we described above marks up what a linguistic expression refers to. But let's say that we want to know the basics for *Question Answering*, namely, the *Who, What, Where, When?* of a sentence. This involves identifying what are called the semantic role labels associated with a verb. What are semantic roles?

- agent: The event participant that is doing or causing the event to occur;
- theme/figure: The event participant who undergoes a change in position or state;
- experiencer: The event participant who experiences or perceives something;
- source: The location or place from which motion begins; The person from whom the theme is given;
- goal: The location or place to which the motion is directed or terminates;
- recipient: The person who comes into possession of the theme;
- patient: The event participant who is affected by the event;
- instrument: The event participant used by the Agent to do or cause the event;
- location/ground: The location or place associated with the event itself.

The resulting annotated data explicitly identifies entity extents and the target relations between the entities.

- [The man]$_{agent}$ painted [the wall]$_{patient}$ with [a paint brush]$_{instrument}$.
- [Mary]$_{figure}$ walked to [the cafe]$_{goal}$ from [her house]$_{source}$.
- [John]$_{agent}$ gave [his mother]$_{recipient}$ [a necklace]$_{theme}$.
- [My brother]$_{theme}$ lives in [Milwaukeel]$_{location}$.

Language Data and Machine Learning

Now that we have reviewed the methodology of language annotation along with some examples of annotation formats over linguistic data, we describe the computational framework within which such annotated corpora are used, namely that of machine learning. Machine learning is the name given to the area of Artificial Intelligence concerned with the development of algorithms which learn or improve their performance from experience or previous encounters with data. They are said to learn (or generate) a function that maps an particular input data to the desired output. For our purposes, the "data" that a machine learning (ML) algorithm encounters is natural language, most often in the form of text, and typically annotated with tags that highlight the specific features that are relevant to the learning task. As we will see, the annotation schemas discussed above, for example, provide rich starting points as the input data source for the machine learning process (the training phase).

When working with annotated datasets in natural language processing, there are typically three major types of machine learning algorithms that are used:

Supervised learning
: Any technique that generates a function mapping from inputs to a fixed set of labels (the desired output). The labels are typically metadata tags provided by humans who annotate the corpus for training purposes.

Unsupervised learning
: Any technique that tries to find structure from a input set of unlabeled data.

Semi-supervised learning
: Any technique that generates a function mapping from inputs of both labeled data and unlabeled data; a combination of both supervised and unsupervised learning.

Classification

Classification is the task of identifying the labeling for a single entity from a set of data. For example, in order to distinguish *spam* from *non-spam* in your email inbox, an algorithm called a classifer is trained on a set of labeled data, where individual emails have been assigned the label [+spam] or [-spam]. It is the presence of certain (known) words or phrases in an email that helps to identify an email as spam. These words are essentially treated as features that the classifier will use to model the positive instances of spam as compared to non-spam. Another example of a classification problem is patient diagnosis, from the presence of known symptoms and other attributes. Here we would identify a patient as having a particular disease, A, and label the patient record as [+disease-A] or [- disease-A], based on specific features from the record or text. This might include blood pressure, weight, gender, age, existence of symptoms, and so forth. The most common algorithms used in classification tasks are: Maximum Entropy (Maxent), Naive Bayes, Decision trees, and Support Vector Machines (SVMs).

Clustering

Clustering is the name given to machine learning algorithms that find natural groupings and patterns from the input data, without any labeling or training at all. The problem is generally viewed as an unsupervised learning task, where the dataset is either unlabeled or the labels are ignored in the process of making clusters. The clusters that are formed are "similar in some respect", and the other clusters formed are "dissimilar to the objects" in other clusters. Some of the more common algorithms used for this task include: K-means, Hierarchical clustering, Kernel Principle Component Analysis, and Fuzzy C-means.

Structured Pattern Induction

Structured pattern induction involves learning more than just the label or category of a single entity, but rather learning a sequence of labels, or other structural dependencies between the labeled items. For example, a sequences of labels might be: a stream of phonemes in a speech signal (in speech recognition); a sequence of part-of-speech tags in a sentence corresponding to a syntactic unit (phrase); a sequence of dialog moves in a phone conversation, and so forth. Algorithms used for such problems include: Hidden Markov Models (HMMs), Conditional Random Fields (CRFs), and Maximum Entropy Markov Models.

We will return to these approaches in more detail when we discuss machine learning in greater depth in Chapter 6 .

The Annotation Development Cycle

The features we use for encoding a specific linguistic phenomenon must be rich enough to capture the desired behavior in the algorithm that we are training. These linguistic descriptions are typically distilled from extensive theoretical modeling of the phenomenon. The descriptions in turn form the basis for the annotation values of the specification language, which are themselves the features used in a development cycle for training and testing an identification or labeling algorithm over text. Finally, based on an analysis and evaluation of the performance of a system, the model of the phenomenon may be revised, for retraining and testing.

We call this particular cycle of development the MATTER methodology (Pustejovsky, 2006):

Model
: Structural descriptions provide theoretically-informed attributes derived from empirical observations over the data;

Annotate
: Annotation scheme assumes a feature set that encodes specific structural descriptions and properties of the input data;

Train
: Algorithm is trained over a corpus annotated with the target feature set;

Test
: Algorithm is tested against held-out data;

Evaluate
: Standardized evaluation of results;

Revise
: Revisit the model and the annotation specification, in order to make the annotation more robust and reliable with use in the algorithm.

We assume that there is some particular problem or phenomenon that has sparked your interest, for which you will need to label natural language data for training for machine learning. Consider two kinds of problems. First imagine a direct text classification task. It might be that you are interested in classifying your email according to its content, or with a particular interest in filtering out spam. Or perhaps you are in-

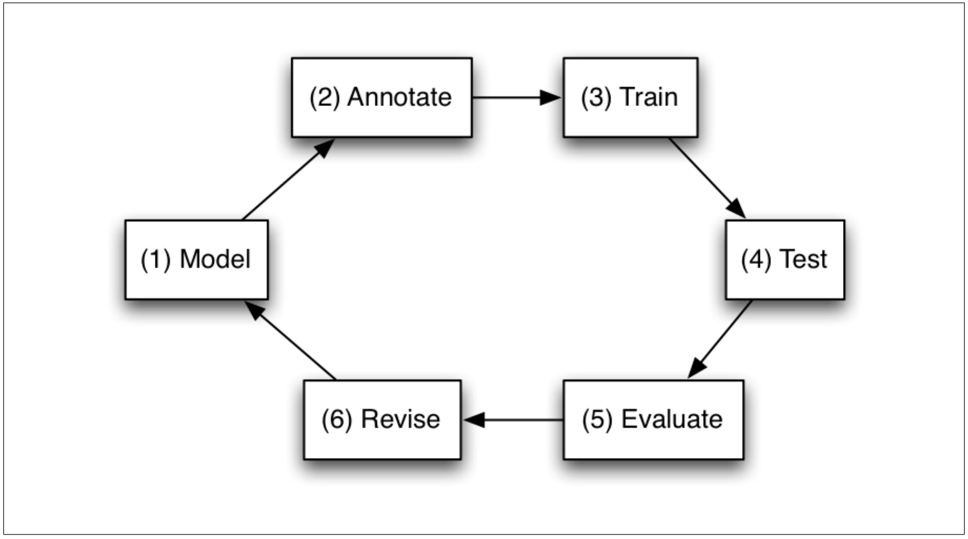


Figure 1-10. The MATTER cycle

terested in rating your incoming mail on a scale of what emotional content is being expressed in the message.

Secondly, let us consider a more involved task, performed over this same email corpus, namely, identifying what are known as **named entities**. These are references to everyday things in our world that have proper names associated with them; e.g., people, countries, products, holidays, companies, sports, religions, etc.

Finally, imagine an even more complicated task, that of identifying all the different events that have been mentioned in your mail (birthdays, parties, concerts, classes, airline reservations, upcoming meetings, etc.). Once this has been done, you will need to "time stamp" them and order them, that is, identify when they happened, if in fact they did. This is called the **temporal awareness** problem, and is one of the most difficult in the field.

We will use these different tasks throughout this section to help us clarify what is involved with the different steps in the annotation development cycle introduced above.

Model the phenomenon

The first step in the MATTER development cycle is "Model the Phenomenon". The steps involved in modeling, however, vary greatly, depending on the nature of the task you have defined for yourself. In this section, we look at what modeling entails and how you know when you have an adequate first approximation of a model for your task.

The parameters associated with creating a model are quite diverse and it is difficult to get different communities to agree on just what a model is. In this section we will be pragmatic and discuss a number of approaches to modeling and show how they provide the basis from which to created annotated datasets. Briefly, a model is a characterization of a certain phenomenon in terms that are more abstract than the elements in the domain being modeled. For the following discussion, we will define a model as consisting of a vocabulary of terms, T, the relations between these terms, R, and their interpretation, I. So, a model, M, can be seen as a triple, $M = <T,R,I>$. To better understand this notion of model, let us consider the scenarios introduced above. For spam detection, we can treat it as a binary text classification task, requiring the simplest model with the categories (terms) *spam* and *not-spam* associated with the entire email document. Hence, our model is simply that given below:

- T = {Document_type, Spam, Not-Spam}
- R = {Document_type ::= Spam | Not-Spam}
- I = {Spam = "something we don't want!", Not-Spam = "something we do want!"}

The document itself is labeled as being a member of one of these categories. This is called document annotation and is the simplest (and most coarse-grained) annotation possible. Now, when we say that the model contains only the label names for the categories (e.g., sports, finance, news, editorials, fashion, etc.), this means that there is no other annotation involved. This does not mean that the content of the files is not subject to further scrutiny, however. A document that is labeled as a category, A, for example, is actually analyzed as a large feature vector containing at least the words in the document. A more fine-grained annotation for the same task would be to identify specific words or phrases in the document and label them as also being associated with the category directly. We'll return to this strategy in Chapter 7. Essentially, the goal of designing a good model of the phenomenon (task) is that this is where you start for designing the features that go into your learning algorithm. The better the features, the better the performance of the machine learning algorithm!

Preparing a corpus with annotations of named entities, as mentioned above, involves a richer model than the spam-filter application just discussed. We introduced a four-category ontology for named entities in the previous section, and this will be basis for our model to identify named entities (NEs) in text. The model is illustrated below:

- T = {Named_Entity, Organization, Person, Place, Time}
- R = {Named_Entity ::= Organization | Person | Place | Time}
- I = {Organization = "list of organizations in a database", Person = "list of people in a database", Place = "list of countries, geographic locations, etc.", Time = "all possible dates on the calendar"}

This model is necessarily more detailed because we are actually annotating spans of natural language text, rather than simply labeling documents (e.g., emails) as spam or not-spam. That is, within the document, we are recognizing mentions of companies, actors, countries, and dates.

Finally, what about an even more involved task, that of recognizing all **temporal information** in a document? That is, *questions like those below:*

- *When* did that meeting take place?
- *How long* was John on vacation?
- Did Jill get promoted *before or after* she went on maternity leave?

We won't go into the full model for this domain, but let's see what is minimally necessary in order to create annotation features to understand such questions. First we need to distinguish between *Time expressions* ("yesterday", "January 27", "Monday"), *Events* ("promoted", "meeting", "vacation"), and *Temporal relations* ("before", "after", "during"). Because our model is so much more detailed, let's divide up the descriptive content by domain.

- Time_Expression ::= TIME | DATE | DURATION | SET
  - —TIME: 10:15 am, 3 o'clock, etc.
  - —DATE: Monday, April, 2011
  - —DURATION: 30 minutes, two years, four days
  - —SET: every hour, every other month
- Event: meeting, vacation, promotion, maternity leave, etc.
- Temporal_Relations ::= BEFORE | AFTER | DURING | EQUAL | OVERLAP | ...

We will come back to this problem in a later chapter, when we discuss the impact of the initial model on the subsequent performance of the algorithms you are trying to train over your labeled data.

In later chapters, we will see that there are actually several models that might be appropriate for describing a phenomenon, each providing a different view of the data. We will call this *multi-model annotation* of the phenomenon. A common scenario for multi-model annotation involves annotators who have domain expertise in an area (such as biomedical knowledge). They are told to identify specific entities, events, atttributes, or facts from documents, given their knowledge and interpretation of a specifica area. From this annotation, non-experts can be used to markup the structural (syntactic) aspects of these same phenomena, thereby making it possible to gain domain expert understanding without

Once you have an initial model for the phenomena associated with the problem task you are trying to solve, you effectively have the first *tag specification*, or *spec*, for the annotation. This is the document from which you will create the blueprint for how to annotate the corpus with the features in the model. This is called the *annotation guideline*, and we talk about this in the next section.

Annotate with the Specification

Now that you have a model of the phenomenon encoded as a specification document, you will need to train human annotators to markup the dataset according to the tags that are of importance to you. This is easier said than done. The *annotation guideline* helps direct the annotators in the task of identifying the elements and then associating the appropriate features with them, when they are identified.

here are essentially two kinds of tags that will concern us when annotating natural language data: *consuming* and *non-consuming* tags. A consuming tag refers to a metadata tag that has real content from the dataset associated with it (e.g., it "consumes" some text); a non-consuming tag, on the other hand, is a metadata tag that is inserted into the file but is not associated with any actual part of the text. An example will help make this distinction clear. Say that we want to annotate text for temporal information, as discussed above. Namely, we want to annotate for three kinds of tags: times (Timexes), temporal relations (TempRels), and events. In the first sentence below, each of these tags is expressed directly as real text. That is, they are all consuming tags ("promoted" is marked as an Event, "before" is marked as aTempRel, and "the summer" is marked as a Timex). Notice, however, that in the second sentence, there is no explicit temporal relation in the text, even though we know that it's something like "on". So, we actually insert a TempRel with the value of "on" in our corpus, but the tag is flagged as a "non-consuming" tag.

- John was [promoted]$_{\text{Event}}$ [before]$_{\text{TempRel}}$ [the summer]$_{\text{Timex}}$.
- John was [promoted]$_{\text{Event}}$ [Monday]$_{\text{Timex}}$.

An important factor when creating an annotated corpus of your text is, of course, consistency in the way the annotators markup the text with the different tags. One of the most seemingly trivial problems is the most problematic when comparing annotations; namely, the extent or the *span of the tag*. Compare the three annotations below. In the first, the Organization tag spans "QBC Productions", leaving out the company identifier "Inc." and the location "of East Anglia", while these are included in varying spans in the next two annotations.

- [QBC Productions]$_{\text{Organization}}$ Inc. of East Anglia
- [QBC Productions Inc.]$_{\text{Organization}}$ of East Anglia
- [QBC Productions Inc. of East Anglia]$_{\text{Organization}}$

Each of these might look correct to an annotator, but only one actually corresponds to the correct markup in the annotation guideline. How are these compared and resolved?

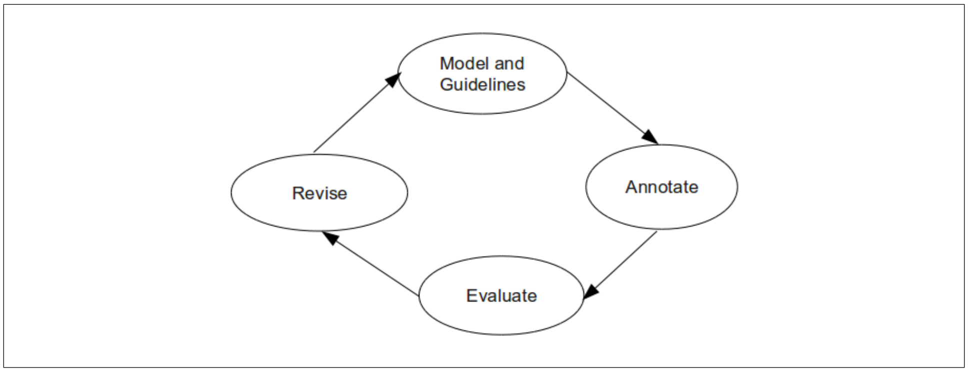


Figure 1-11. Annotation model

In order to assess how well an annotation task is defined, we use *Inter-Annotator Agreement* (IAA) scores to how individual annotators compare to each other. If an IAA score is high, that is an indication that the task is well-defined and other annotators will be able to continue the work. This is typically defined using a statistical measure called a *Kappa Statisic*. For comparing two annotations against each other, the *Cohen Kappa* is usually used, while when comparing more than two annotations, a *Fleiss Kappa* measure is used. These will be defined in Chapter 8.

Note that having a high IAA score doesn't necessarily mean that the annotations are correct; it simply means that the annotators are all interpreting your instructions consistantly in the same way. Your task may still need to be revised even if your IAA scores are high. This will be discussed further in Chapter 9.

Once you have your corpus annotated by at least two people (more is preferable, but not always practical), it's time to create the *gold standard corpus*. The gold standard is the final version of your annotated data. It uses the most up-to-date specification that you created during the annotation process, and it has everything tagged correctly according to the most recent guidelines. This is the corpus that you will use for machine learning, and it is created through the process of *adjudication*. At this point in the process you (or someone equally familiar with all the tasks) will compare the annotations and determine which tags in the annotations are correct and should be included in the gold standard.

Train and Test the algorithms over the corpus

Now that you have adjudicated your corpus, you can use your newly-created gold standard for machine learning. The most common way to do this is to divide your corpus into two parts: the *development corpus* and the *test corpus*. The development corpus is then further divided into two parts: the *training set* and the *development-test*

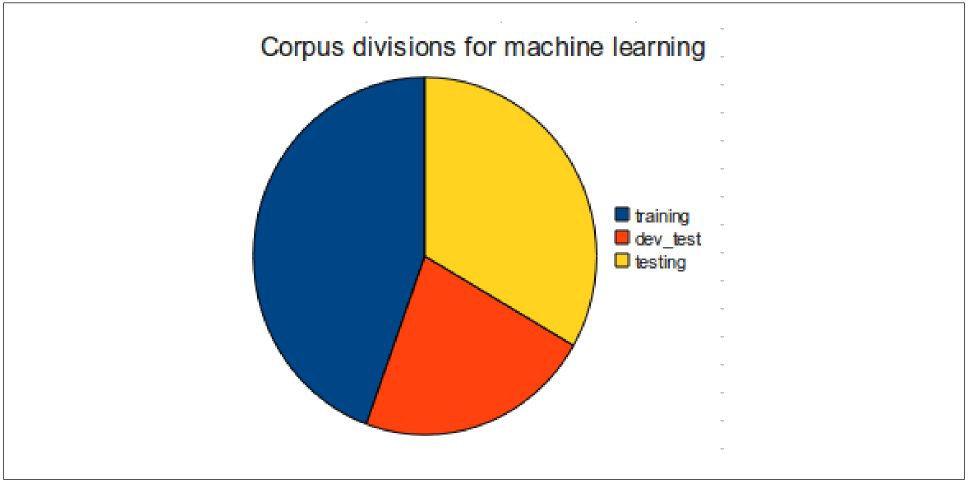


Figure 1-12. Corpus divisions for machine learning

set. Figure Figure 1-12 shows a standard breakdown of a corpus, though different distributions might be used for different tasks. The files are normally distributed randomly into the different sets.

The training set is used to train the algorithm that you will use for your task. The development-test (dev-test) set is used for error analysis. Once the algorithm is trained, it is run on the dev-test set and a list of errors can be generated to find where the algorithm is failing to correctly label the corpus. Once sources of error are found, the algorithm can be adjusted and re-trained, then tested against the dev-test set again. This procedure can be repeated until satisfactory results are obtained.

Once the training portion is completed, the algorithm is run against the held-out test corpus, which until this porint has not been involved in traning or dev-testing. By holding the data out we can show how well the algorithm will perform on new data, which gives an expectation of how it would perform on data that someone else creates as well.

Evaluate the results

The most common method for evaluating the performance of your algorithm is to calculate how accurately it labels your dataset. This can be done by measuring the fraction of the results from the dataset that are labeled correctly using a standard technique of "relevance judgment" called the *Precision and Recall Metric*.

Here's how it works. For each label you are using to identify elements in the data, the dataset is divided into two subsets, one that is labeled "relevant" to the label, and one which is not relevant. *Precision* is a metric that is computed as the fraction of the correct instances from those that the algorithm labeled as being in the relevant subset. *Recall*

- PRECISION: $P = \frac{tp}{tp + fp}$
- RECALL: $R = \frac{tp}{tp + fn}$
- ACCURACY: $A = \frac{tp + tn}{tp + tn + fp + fn}$

Figure 1-13. Precision and Recall equations

is computed as the fraction of correct items among those that actually belong to the relevant subset. The following confusion matrix helps illustrate how this works:

		Predicted	**Labeling**
		positive	negative
Gold	postive	true positive (tp)	false negative (fn)
Labeling	negative	false positive (fp)	true negative (tn)

Given this table, we can define both precision and recall as follows, along with a conventional definition of *accuracy*.

The values of P and R are typically combined into a single metric called the *F-measure*, which is the harmonic mean of the two.

F = 2*((P*R)/(P+R))

This creates an overall score used for evaluation where precision and recall are measured equally, though depending on the purpose of your corpus and algorithm, a variation of this measure, such as one that rates precision higher than recall, may be more useful to you. We will give more detail about how these equations are used for evaluation in Chapter 8.

Revise the Model and Algorithms

Once you have evaluated the results of training and testing your algorithm on the data, you will want to do an error analysis to see where it performed well and where it made mistakes. This can be done with various packages, which we will discuss in Chapter 8, including the creation of what are called confusion matrices. These will help you go back to the design of the model, in order to create better tags and features that will subsequently improve your gold standard, and consequently result in better performance of your learning algorithm.

A brief example of model revision will help make this point. Recall the model for named entity extraction from the previous section, where we distinguished between four types of entities: *organizations*, *places*, *times*, and *persons*. Depending on the corpus you have assembled, it might be the case that you are missing a major category, or that you would be better off making some subclassifications within one of the existing tags. For example, you may find that the annotators are having a hard time knowing what to do with named occurrences or events, such as Easter, 9-11, or Thanksgiving. These denote more than simply Times, and suggest that perhaps a new category should be added to the model, i.e., Event. Additionally, it might be the case that there is reason to distinguish geopolitical Places from non-geopolitical Places. As with the "Model-Annotate" and "Train-Test"" cycles, once such additions and modifications are made to the model, the MATTER cycle begins all over again, where revisions will typically bring improved performance.

Summary

- Natural language annotation is an important step in the process of training computers to undersand human speech for tasks such as question answering, machine translation, and summarization.
- All of the layers of linguistic research, from phonetics to semantics to discourse analysis, are used in different combinations for different machine learning tasks.
- In order for annotation to provide statistically useful results, it must be done on a sufficiently large dataset, called a *corpus*. The study of language using corpora is *corpus linguistics*.
- Corpus linguistics began in the 1940s, but did not become a feasible way to study language until decades later when the technology caught up to the demands of the theory.
- A corpus is a collection of machine-readable texts that are representative of natural human language. Good corpora are *representative* and *balanced* with respect to the genre or language that they seek to represent.
- The uses of computers with corpora have developed over the years from simple keyword-in-context (KWIC) indexes and concordances that allowed full-text documents to be searched easily, to modern statistically-based machine learning techniques.
- Annotation is the process of augmenting a corpus with higher-level information, such as part of speech tagging, syntactic bracketing, anaphora resolution, word senses, etc. Adding this information to a corpus allows the computer to find features that can make a defined task easier and more accurate.
- Once a corpus is annotated, the data can be used in conjunction with machine learning algorithms that perform classification, clustering and pattern induction tasks.
- Having a good annotation scheme and accurate annotations are critical for machine learning that relies on data outside of the text itself. The process of developing the

annotated corpus is often cyclical, with changes made to the tagsets and tasks as the data is studied further.

- Here we refer to the annotation development cycle as the MATTER cycle--Model, Annotate, Train, Test, Evaluate, Revise.
- Often before reaching the Testing step of the process, the annotation scheme has already gone through several revisions of the Model and Annotate stages.
- This book will show you how to create an accurate and effective annotation scheme for a task of your choosing, apply the scheme to your corpus, and then use machine learning techniques to train a computer to perform the task you designed.

CHAPTER 2

Defining Your Goal and Dataset

Creating a clear definition of your annotation goal is vital for any project aiming to incorporate machine learning. When you are designing your annotation tagsets, writing guidelines, working with annotators, and training algorithms, it can be easy to become sidetracked by details and lose sight of what you want to achieve. Having a clear goal to refer back to can help, and in this chapter we will go over what you need to create a good definition of your goal, and discuss how your goal can influence your dataset. In particular, we will look at:

- What makes a good annotation goal
- Where to find related research
- How your dataset reflects your annotation goals
- Preparing the data for annotators to use
- How much data you will need for your task

What you should be able to take away from this chapter is a clear answer to the questions "What am I trying to do?", "How am I trying to do it?", and "Which resources best fit my needs?" As you progress through the MATTER cycle the answers to these questions will probably change—corpus creation is an iterative process—but having a stated goal will help keep you from getting off track.

Defining a goal

In terms of the MATTER cycle, at this point we're right at the start of "M"—being able to clearly explain what you hope to accomplish with your corpus is the first step in creating your model. While many of you probably already have a good idea about what you want to do, in this section we'll give you some pointers on how to create a goal definition that is useful and will help keep you focused in the later stages of the MATTER cycle.

We have found it useful to split the goal definition into two steps: first, write a statement of purpose that covers the very basics of your task, and second, using that sentence to expand on the "how"s of your goal. In the rest of this section, we'll give some pointers about how to make sure each of these parts will help you with your corpus task.

The Statement of Purpose

At this point we're assuming that most of you already have some question pertaining to natural language that you want to explore.[1]

But how clearly can you explain what you intend to do? If you can't come up with a one– or two–sentence summary describing what your intended line of research is, then you're going to have a very hard time with the rest of this task. Keep in mind that we are not talking about a sentence like "Genres are interesting"—that's an opinion, not a starting point for an annotation task. Instead, try to have a statement more like this:

> I want to use keywords to detect the genre of a newspaper article in order to create databases of categorized texts.

This statement still going to need a lot of refinement before it can be turned into an annotation model, but it answers the basic questions. Specifically, it says:

- What the annotation will be used for (databases)
- What the overall outcome of the annotation will be (genre classification)
- Where the corpus will come from (news articles)
- How the outcome will be achieved (keywords)

Be aware that you may have a task in mind that will require multiple annotation efforts. For example: say that you're interested in exploring humor. Even if you have the time, money, and people to do a comprehensive study of all aspects of humor, you will still need to break the task down into manageable segments in order create annotation tasks for different types of humor—if you want to look at the effects of sarcasm and puns, you will likely need different vocabularies for each task, or be willing to spend the time to create one over-arching annotation spec that will encompass all annotations, but we suggest that you start small and then merge annotations later if possible.

If it's the case that you do have a broad task that you will need to break down into large subtasks, then make that clear in your summary: "I want to create a program that can generate jokes in text and audio formats, including puns, sarcasm, and exaggeration." Each of the items in that list will require a separate annotation and machine learning task—grouping them together, at least at first, would create such a massively complicated task that it would be difficult to complete, let alone learn from.

To provide some more context, here are a few examples of one-sentence summaries of existing annotation projects and corpora:

1. If you don't really have a project in mind yet, check the Appendices for lists of existing corpora, and read the proceedings from related conferences to see if there's anything that catches your eye, or consider participating in an NLP challenge.

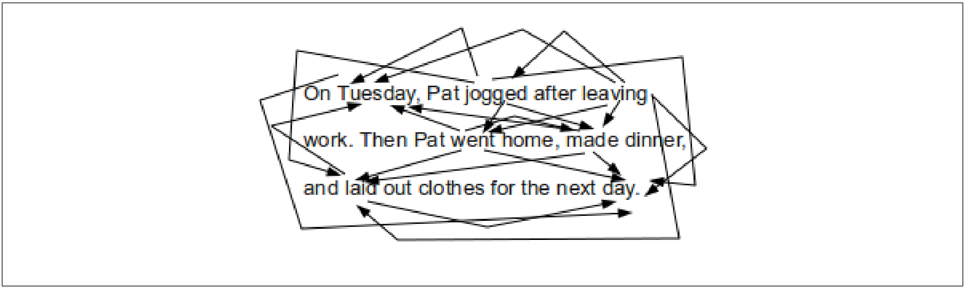


Figure 2-1. All temporal relations over events and times

PropBank	A corpus that annotates verbal propositions and their arguments for examining semantic roles.
British National Corpus	Part-of-speech tagging of a representative sample of British writing and speech for use as a syntactic resource.
Penn Discourse TreeBank	Annotating discourse relations between eventualities and propositions in newswire for learning about discourse in natural language.
MPQA Opinion Corpus	Annotating opinions for use in evalutating emotional language.
TimeBank	Labeling times, events, and their relationships in news texts for use in temporal reasoning.
i2b2 2008 challenge, task 1C	Identifying patient smoking status from medical records for use in medical studies..

These are high-level descriptions of what these corpora are about, but they answer the basic questions necessary for moving forward with an annotation task. In the next section, we'll look at how to take this one sentence and take the first steps towards turn it in to an annotation model.

Refining your Goal: Informativity versus Correctness

Now that you have a statement of purpose for your corpus, you need to turn it into a task description that can be used to create your model—that is, your corpus, annotation scheme, and guidelines.

There is a fine line in annotating corpora between having an annotation that will be the most useful for your task (having high informativity), and having an annotation that is not too difficult for annotators to complete accurately (which results in high levels of correctness).

A clear example of where this tradeoff comes into play is temporal annotation. Imagine that you want to capture all the relationships between times and events in this simple narrative:

> On Tuesday, Pat jogged after leaving work. Then Pat went home, made dinner, and laid out clothes for the next day.

However, Figure 2-1 shows what it would look like to actually create *all* of those relationships, and you can see that there needs to be a large number of connections in order to capture all the relationships between events and times. In such tasks, the number of relationships is almost quadratic to the number of times and events (here we have 21, because the links only go in one direction—if we captured both directions there would be 42 links: x * (x-1)). It wouldn't be very practical to ask an annotator to do all that work by hand; such a task would take an incredibly long time to complete, and an annotation that complex will lead to a lot of errors and omissions; in other words, low correctness. However, asking for a limited set of relations may lead to lower levels of informativity, especially if your annotation guidelines are not very carefully written. We'll discuss annotation guidelines further in (to come).

You have probably realized that with this particular example it's not necessary to have a human create *all* those links—if A occurs before B and B occurs before C, a computer could use *closure rules* to determine that A occurs before C, and annotators would not have to capture that information themselves. It's always a good idea to consider what parts of your task could be done automatically, especially when it can make your annotator's jobs easier without necessarily sacrificing accuracy.

The considerations surrounding informativity and correctness are very much intertwined with one of the biggest factors affecting your annotation task: the *scope* of your project. There are two main aspects of the project scope that you need to consider: (1) how far-reaching the goal is (the scope of the annotation), and (2) how much of your chosen field you plan to cover (the scope of the corpus). We have already touched on (2) a little in the previous section, and we will have more to say about it later, so for now let's look at (1).

The scope of the annotation task

At this point you have already begun to address the question of your task's scope by answering the four questions from the previous section—at the very least you've narrowed down what category of features you'll be using (by answering the "means by which the goal will be achieved" question), and what the overall goal of the annotation will be. However, having a general class that your task can be slotted in to may still leave you with a lot of variables that you will need to consider.

As always, remember that the MATTER cycle is, in fact, a cycle, and as you go through the steps you may find new information that cause your scope to widen or shrink.

It's a bit difficult to discuss scope in general terms, so let's look at some specific examples, and then see how the principles can be applied to other projects. In the temporal relation annotation task discussed previously, the scope of the project has to do with exactly what relations are important to the annotation. Are only the main events in each sentence and their relationships important? Do you want to be able to capture only relations inside of a sentence, or do you want to capture relations between sentence as well? Maybe you only care about the events that have clear temporal anchors, such as "Jay ran on Sunday." Do you think it's important to differentiate between different types of links?

Similar questions can be asked about the newspaper genre classification example. In that task, the relevant question is "How specific do you want your categories to be?" Is it enough to divide articles into broad categories, like "news" and "sports", or do you want a more detailed system that specifies "news:global" and "sports:baseball" (or even "sports:baseball:Yankees")?

If this is your first excursion into annotation and corpus building, start with broader categories or simpler tasks—as you learn the ins and outs of your dataset and annotation tags, you'll be able to refine your project in more meaningful and useful ways.

As you can see, the questions being asked about the two examples so far essentially become questions about classification—in the newspaper example this is a much more obvious correlation, but even the temporal relation example touches on this subject. By defining different possible categories of relationships (inter- and intra-sentential, main verbs versus other verbs) it's much easier to identify what parts you feel will be most relevant to your task.

This book is about annotation, and it's necessarily true that if you have an annotation task, classification is some part of your task. This could involve document-level tags, as in the newspaper example, labels that are associated with each word (or a subset of words), as in a part-of-speech tagging task, or labeling relationships between existing tags. If you can think of your annotation task in terms of a classification task, then that will provide a stable framework for you to start considering the scope of your task. You most likely already have intuitions about what the relevant features of your data and task are, so use those (at least at first) to determine the scope of your task. This intuition can also help you determine what level of informativity you will need for good classification results, and how accurate you can expect an annotator to be.

Linguistic intuition is an extremely useful way to get started thinking about a topic, but it can be misleading and even wrong. Once you've started gathering texts and doing some annotation, if you find that the data do not match your expectations then don't hesitate to re-evaluate your approach.

Let's go back to our four questions from the previous section and see how informativity and correctness come into effect when elaborating on these aspects of your annotation task. Now that you have a better idea of the scope of your project, it should be fairly easy to see what sorts of things you will want to take into account when answering these questions. (Notice that we didn't say that the questions would be *easy* to answer —the trade-off between informativity and correctness is a consideration at all levels, and can make it difficult to decide where to draw the line on a project.)

What will the annotation be used for?

It's likely that the answer to this question hasn't really changed based on the discussion questions we've provided so far—the end product of the annotation, training, and testing is why you're taking on this project in the first place, and is what ultimately informs the answers to the rest of the questions and the project as a whole. However, it's helpful to remind yourself of what you're trying to do before you start expanding your answers to the other questions.

If you have some idea of what machine learning techniques you'll be using, then that can be a consideration as well at this point, but it's not required, especially if this is your first turn around the MATTER cycle.

What will the overall outcome be?

Thinking about the scope of your project in terms of a classification task, now is the time to start describing the outcome in terms of specific categories. Instead of saying "classify newspaper articles into genres", try to decide on the number of genres that you think you'll need to encompass all the types of articles that you're interested in.

For the temporal relation annotation, accuracy versus informativity is a huge consideration, for the reasons described above (go back and look at Figure 2-1 for a refresher on how complicated this task can be). In this case, and in the case of tasks of similar complexity, the specifics of how detailed your task will be will almost certainly have to be worked out through multiple iterations of annotation and evaluation.

For both of these tasks, taking into consideration the desired outcome will help determine the answer to this question. In the genre case, the use of the database is the main consideration point—who will be using it, and for what? Temporal annotation can be used for a number of things, such as summarization, timeline creation, question answering, and so on. The granularity of the task will also inform what needs to be captured in the annotation: if, for example, you are only interested in summarizing the major events in a text, then it might be sufficient to only annotate the relationships of the main event in each sentence.

Where will the corpus come from?

Now that you've thought about the scope of your task, it should be much easier to answer more specific questions about the scope of your corpus. Specifically, it's time to start thinking about the distribution of sources; that is, exactly where all your data will come from and how different aspects of it will be balanced.

Returning to our newspaper classification example, consider whether different news venues have sufficiently different styles that you would need to train an algorithm over all of them, or if a single source is similar enough to the others for the algorithm to be easily generalizable. Are the writing and topics in the New York Times similar enough to the Wall Street Journal that you don't need examples from each? What about newspapers and magazines that publish exclusively online? Do you consider blogs to be news sources? Will you include only written articles, or will you also include transcripts of broadcasts?

For temporal annotation, our experience has been that different publication styles and genres can have a huge impact on how times are used in text (consider a children's story compared to a newspaper article, and how linear (or not) the narration in each tends to be). If you want your end product to cover all types of sources, this might be another splitting point for your tasks—you may need to have different annotation guidelines for different narrative genres, so consider carefully how far-reaching you want your task to be.

Clearly, these considerations are tied in to the scope of your task—the bigger the scope, the more sources you will need to include in your annotation in order to maximize informativity. However, the more disparate sources you include, the more likely it is that you will need to consider having slightly different annotation tasks for each source, which could lower correctness if the tasks are not fully thought out for each genre.

It's also a good idea to check out existing corpora for texts that might be suitable for the task that you are working on. Using a pre-assembled corpus (or a subset of one) has the obvious benefit of lessening the work that you will have to do, but it also means that you will have access to any other annotations that have been done on those files. See Section 2 for more information on linguistic resources.

Don't get too caught up in the beginning with creating the perfect corpus —remember that this process is cyclical, and if you find you're missing something you can always go back and add it later.

How will the result be achieved?

In Chapter 1 we discussed the levels of linguistics—phonology, syntax, semantics, etc. —and gave examples of annotation tasks for each of those levels. Consider at this point, if you haven't already, which of these levels your task fits into. However, don't try to force your task to only deal with a single linguistic level! Annotations and corpora do not always neatly fit into one category or another, and the same is probably true of your own task.

For example, while the temporal relation task that we have been using as an example so far fits fairly solidly into the discourse and text structure level, it relies on having events and times already annotated. But what is an event? Often events are verbs ("He *ran* down the street.") but they can also be nouns ("The *election* was fiercely contested.") or even adjectives, depending whether they represent a state that has changed ("The volcano was *dormant* for centuries before the eruption."). But labeling events is not a purely syntactic task, because (1) not all nouns, verbs, and adjectives are events, and (2) because the context that a word is used will determine whether a word is an event or not. Consider "The *party* lasted until 10" versus "The political *party* solicited funds for the campaign." These examples add a semantic component to the event annotation.

It's very likely that your own task will benefit from bringing in information from different levels of linguistics. Part-of-speech tagging is the most obvious example of additional information that can have a huge impact on how well an algorithm performs an NLP task: knowing the part of speech of a word can help with word sense disambiguation ("call the police" versus "police the neighborhood"), determining how the syllables of a word are pronounced (consider the verb pre*sent* versus the noun *pre*sent —this is a common pattern in American English), and so on.

Of course, there is always a trade-off: the more levels (or partial levels—it might not be necessary to have part-of-speech labels for all your data, they might only be used on words that are determined to be interesting some other way) that your annotation includes, the more informative it's likely to be. But the other side of that is the more complex your task is, the more likely it is that your annotators will become confused, thereby lowering your accuracy.

Background research

Now that you've considered what linguistic levels are appropriate for your task, it's time to do some research into related work. Creating an annotated corpus can take a lot of effort, and while it's not impossible to create a good annotation task completely on your own,checking the state of the industry can save you a lot of time and effort—chances are that there's some research that's relevant to what you've been doing, and it helps to not have to re-invent the wheel.

For example, if you are interested in temporal annotation, you know by now that ISO-TimeML is the ISO standard for time and event annotation, including temporal relationships. But this fact doesn't require that all temporal annotations use the ISO-TimeML schema as-is. Different domains, such as medical and biomedical text analysis, have found that TimeML is a useful starting place, but in some cases provides too many options for annotators, or in other cases, does not cover a particular case relevant to the area being explored. Looking at what other people have done with existing annotation schemes, particularly in fields related to those you are planning to annotate, can make your own annotation task much easier to plan.

While the library and, of course, Google usually provides a good starting place, those sources might not have the latest information on annotation projects, particularly because the primary publishing grounds in computational linguistics are conferences and their related workshops. In the following sections we'll give you some pointers to organizations and workshops that may prove useful.

Language Resources

Currently there are two primary resources for pre-assembled corpora. The Linguistic Data Consortium (LDC), which has a collection of hundreds of corpora in both text and speech, from a variety of languages. While most of the corpora are available to non-members (sometimes for a fee), some of them do require LDC membership. The LDC is run by the University of Pennsylvania, and details about membership and the cost of corpora are available on their website *http://www.ldc.upenn.edu*.

The European Language Resources Association is another repository of both spoken and written corpora from many different languages. As with the LDC, it is possible to become a member of the ELRA in order to gain access to the entire database, or access to individual corpora can be sought. More information is available at their website, *http://ELRA.info*.

With both the LDC and the ELRA, it's possible that while you would need to pay to gain access to the most up-to-date version of a corpus, an older version may be available for download from the group that created the corpus, so it's worth checking around for availability options if you are short of funds. And, of course, check the license on any corpus you plan to use to ensure that it's available for your purposes, no matter where you obtain it from.

Organizations and Conferences

Much of the work on annotation that is available to the public is being done at universities, making conference proceedings the best place to start looking for information about tasks that might be related to your own. Below is a list of some of the bigger conferences that examine annotation and corpora, as well as some organizations that are interested in the same topic:

- Association for Computational Linguistics (ACL)
- Institute of Electrical and Electronics Engineer (IEEE)
- Language Resources and Evaluation Conference (LREC)
- European Language Resources Association (ELRA)
- Conference on Computational Linguistics (COLING)
- American Medical Informatics Association (AMIA)

The Linguist List (linguistlist.org) is not an organization that sponsors conferences and workshops itself, but it does keep an excellent up-to-date list of calls for papers and dates of upcoming conferences. It also provides a list of linguistic organizations that can be sorted by linguistic level.

NLP Challenges

In the past few years, NLP challenges hosted through conference workshops have become increasingly common. These challenges usually present a linguistic problem, a training and testing dataset, and a limited amount of time during which teams or individuals can attempt to create an algorithm or rule set that can achieve good results on the test data.

The topics of these challenges vary widely, from part-of-speech tagging to word sense disambiguation to text analysis over biomedical data, and they are not limited to English. Some workshops that you may want to look into are:

- SemEval - a workshop held every 3 years as part of the Association for Computational Linguistics, involves a variety of challenges including word sense disamibiguation, temporal and spatial reasoning, and machine translation.
- Conference on Natural Language Learning (CoNLL) Shared Task - a yearly NLP challenge held as part of the Special Interest Group on Natural Language Learning of the Association for Computational Linguistics. Each year a new NLP task is chosen for the challenge. Past challenges include uncertainty detection, extracting syntactic and semantic dependencies, and multi-lingual processing.
- i2b2 NLP Shared Tasks - The i2b2 group is focused on using NLP in the medical domain, and each year holds a challenge invoving reasoing over patient documents. Past challenges have focused on co-morbitidy, smoking status, and identifying medication information.

If you would like to be involved in a machine learning task but don't want to necessarily create a dataset and annotation yourself, signing up for one of these challenges is an excellent way to get involved in the NLP community. NLP challenges are also useful in that they are a good reference for tasks that might not have a lot of time or funding. However, it should be noted that the time constraints on NLP challenges often mean that the obtained results are not the best possible overall, simply the best possible given the time and data.

Assembling your dataset

We've already discussed some aspects that you will need to consider when assembling your dataset: the scope of your task, whether existing corpora contain documents and annotations that would be useful to you, and how varied your sources will be.

If you are planning on making your dataset public, make very sure that the information you have permission to re-distribute the information that you are annotating. In some cases it is possible to release only the stand-off annotations and a piece of code that will collect the data from websites, but it's best and easiest to simply ask permission of the content provider. Particularly if your corpus and annotation is for business rather than purely educational purposes.

Collecting data from the Internet

If you are doing textual annotation, you will probably be collecting your corpus from the Internet. There are a number of excellent books that will provide specifics for how to gather URLs and string HTML tags from websites: the NLTK book (Bird, Klein, and Loper, 2010) provides details on how to format web pages with the Natural Language Tool Kit, and other books provide information on mining twitter streams, forums, and other Internet resources.

Eliciting data from people

So far we have assumed that you will be annotating texts or recordings that already exist. But for some tasks, the data just aren't there, or at least don't exist in a form that's going to be of any use.

This applies, we think, more to tasks requiring annotation of spoken or visual phenomena than written work—unless you are looking for something very particular when it comes to text, it's rarely necessary to have people generate writing samples for you. However, it's very common to need spoken samples, or recordings of people performing particular actions for speech or motion recognition projects.

If you *do* need to elicit data from humans and are affiliated with a university or business, you will probably have to seek permission from lawyers, or even an Internal Review Board (IRB). Even if you are doing your own research project, be very clear with your volunteers about what you're asking them to do and why you're asking them to do it.

When it comes to eliciting data (as opposed to just collecting it), there are a few things you need to consider: in particular, do you want your data to be spontaneous, or read? Do you want each person to say the same thing, or not? Let's take a look at what some of the differences are, and how it might affect your data.

Read speech

"Read speech" means that, while collecting data, you have each person read the same set of sentences or words. If, for example, you wanted to compare different dialects or accents, or train a speech recognition program to detect when people are saying the same thing, then this is probably the paradigm that you will want to use.

The VoxForge corpus uses this method—it provides a series of prompts that speakers can record on their own and submit with a user profile describing their language background.

If you do decide to have people read text from a prompt, be aware that how the text is presented (font, bold, italics) can have large effects on how the text is read. You may need to do some testing to make sure your readers are giving you useful sound bites.

Recordings of news broadcasts can also be considered "read speech", but be careful—the cadence of news anchors is often very different from "standard" speech, so these recordings might not be useful, depending on your goal.

A detailed description of the data collection process for the WSJCAM0 Corpus can be found here: *http://www.ldc.upenn.edu/Catalog/readme_files/wsjcam0/wsjcam0.html*

Spontaneous speech

Naturally, spontaneous speech is collected without telling people what to say. This can be done by asking people open-ended questions and recording their responses, or simply recording conversations (with permission, of course!).

Preparing your data for annotation

Data preparation falls into two main tasks: the more theoretical act of deciding how to present your documents to your annotators, and very practical matter of coercing those documents into a form that will work with the annotation tool that you choose. The discussion of annotation tools will take place in (to come), so for now we'll just focus on deciding what information to present to your annotators.

Metadata

Deciding what information to give to your annotators can be trickier than it seems. You have the text or speech files that you want annotated, but how much do you want your annotators to know about the source of those files? For example, where they came from and who produced them?

This may seem like a trivial problem, but information about a document can influence annotators in sometimes unexpected ways. Opinion annotation in particular can be susceptible to this problem. An obvious example would be, in examining the polarity of movie reviews, informing the annotators of the star rating that the reviewer gave the movie. If you want an annotator to look for both positive and negative adjectives, informing them that the reviewer gave the movie 10 stars can prime them to look only for positive adjectives.

Somewhat similarly, if annotators are examining newspaper or Wikipedia articles for factuality, informing them of the source of the article, or if the Wikipedia page was flagged for being biased could influence the annotator, particularly if a source is, to them, associated with biased reporting, or if they tend to believe that Wikipedia flags are accurate.

Another example is If you are working on creating a phonetic transcription of recorded data, informing your annotators of the birthplace of a speaker could potentially bias them to hear pronunciations in a particular way, especially if they are new to phonetic transcription.

Bias can appear in unexpected ways—it's useful to try a few variations on the data presentation to eliminate different biasing factors.

Pre-processed data

Another consideration in data presentation is giving your annotators data that already has some information marked up.

There are two things to consider before doing that, however:

- Is this information necessary for a human annotator, or would it be more useful for an algorithm?
- Is this information going to be correct, or is it something that the annotator will have to fix/examine for errors?

Part-of-speech is a great example of information that would be useful for a machine to have, but that might not be necessary to give to an annotator (especially if the annotator is a native speaker of the language he or she is evaluating). Presenting an annotator with too much information can make the task confusing, and lower accuracy without necessarily increasing informativity. Of course, what information is useful will greatly depend on who your annotators are, so this consideration should be revisited after you find annotators.

Whether or not your preprocessed information is correct will have a huge impact on your annotation project. Certainly, it saves time to do part of your annotation automatically if the tools exist, but people in general tend to like simple instructions, and so "Label all the named entities/time/events/etc, then create links between them", might be an easier rule to follow (especially since it could be done as two different tasks) than "We have marked up most of the entities/times/events for you, but they might not be correct so we'd like you to fix any errors while creating links."

In our experience, asking annotators to correct partial annotations generally leads to mistakes not being fixed, especially when combined with other instructions at the same time. Splitting the annotation into related tasks, such as correcting the generated labels and then later on creating links is probably the best way to deal with this. It is more time-consuming, but will result in higher-quality annotations.

The size of your corpus

Now that you know what kind of data you're looking for and how you're going to present it, you have to decide how much data you're actually going to collect and annotate. If you're planning on using a corpus that already exists than the overall size of the corpus is decided for you, but you still might have to determine how much of that corpus you want to annotate.

Generally speaking, no matter what your annotation goal is, the more data you collect and annotate, the closer to achieving that goal you'll be able to get. However, most of the time "bigger is better" isn't a very practical mantra when discussing annotation tasks—time, money, limited resources, and attention span are all factors that can limit how much annotation you and your annotators can complete.

If this is your first pass at data collecting, the most important thing is to have a sample corpus that has examples of all the phenomena that you are expecting to be relevant to your task.

That being said, we recommend starting small when it comes to your first attempts at annotating documents—select a handful documents for your annotators first, and see how well your annotation task and guidelines work (annotation guidelines will be discussed further in (to come)). Once you have some of the problems worked out, then you can go back and add to your corpus as needed.

Unfortunately, there's no magic number that we can give you to decide how big your corpus will need to be in order to get good results. How big your corpus needs to be will depend largely on how complex your annotation task is, but even having a way to quantify "complexity" in an annotation scheme won't solve all the problems. However, corpora that are in use can provide a rule of thumb for how big you can expect your own corpus to be.

Existing Corpora

A rule of thumb for gauging how big your corpus may need to be is to examine existing corpora that are being used for tasks similar to yours. Table 2-1 shows sizes of some of the different corpora that we have been discussing so far. As you can see, they do not all use the same metric for evaluating size. This is largely a function of the purpose of the corpus—corpora designed for evaluation at a document level, such as the movie review corpus included in NLTK, will provide the number of documents as a reference, while annotation tasks that are done at a word- or phrase-level will report on the number of words or tokens for their metric.

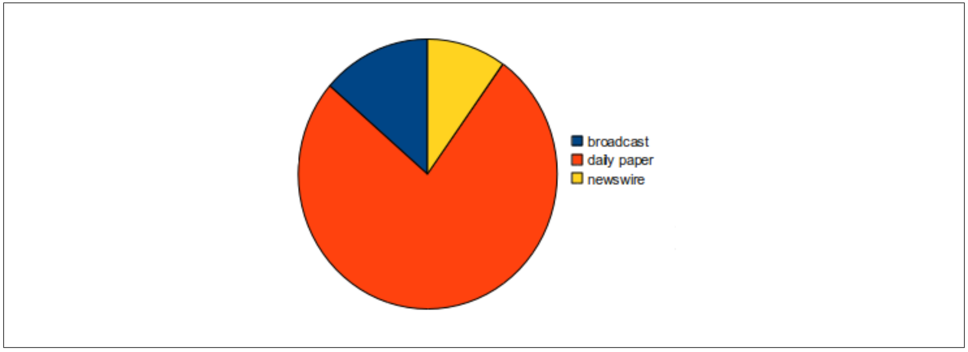


Figure 2-2. Production Circumstances in TimeBank

Table 2-1. Sizes of existing corpora

Penn Discourse Tree Bank	1 million words
British National Corpus	100 million words
American National Corpus	22 million words (so far)
TimeBank 1.2	183 documents
i2b2 2008 challenge—smoking status	502 hospital discharge summaries
TempEval 2 (part of SemEval 2010)	10K to 60K tokens per language dataset
Disambiguating Sentiment Ambiguous Adjectives (Chinese language data, part of SemEval 2010)	4,000 sentences

You will notice that the size of the last three corpora are generally smaller than the other corpora listed—this is because those three were used in NLP challenges as part of existing workshops, and part of the challenges is to perform an NLP machine learning task in a limited amount of time. This limit includes the time spent creating the training and testing datasets, and so the corpora have to be much smaller in order to be feasible to annotate, and in some cases the annotation schemes are simplified as well. However, results from these challenges are often not as good as they would be if more time could have been put into creating larger and more thuroughly annotated datasets.

Distributions within corpora

We have previously discussed including different types of sources in your corpus in order to increase informativity. Here we will show examples of some of the source distributions in existing corpora.

For example, TimeBank is a selection of 183 news articles that have been annotated with time and event information. However, not all the articles in TimeBank were produced the same way: some are broadcast transcripts, some are articles from a daily newspaper, and some were written for broadcast over the newswire. The breakdown of this distribution is shown in figure (REF).

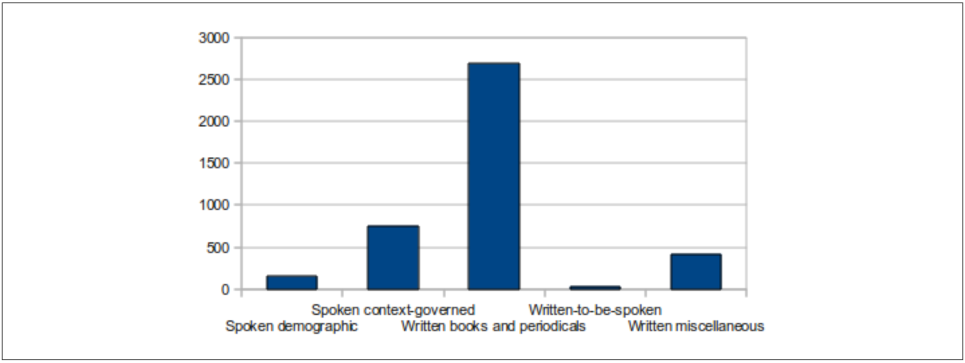


Figure 2-3. Distribution of Text Types in the BNC

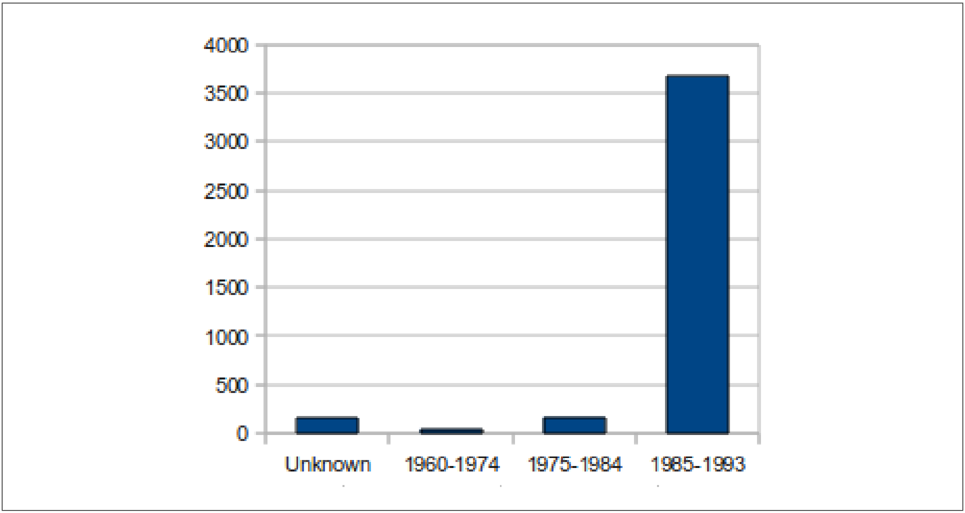


Figure 2-4. Publication Dates in the BNC

As you can see, while the corpus trends heavily towards daily published newspapers, other sources are also represented. Having those different sources has provided insight into how time and events are reported in similar, but not identical, media.

The British National Corpus (BNC) is another example of a corpus that draws from many sources—sources even more disparate than those in TimeBank. The graph below shows the breakdown of text types in the BNC, as described in the Reference Guide for the BNC.

Naturally, there are other distribution aspects that can be taken into consideration when evaluating how balanced a corpus is. The BNC also provides analysis of their corpus based on publication dates, domain, medium, as well as analysis of the sub-groups, including information about authors and intended audiences.

Naturally, for your own corpus it's unlikely that you will need to be concerned with having representative samples of all of these possible segmentations. That being said, minimalizing the number of factors that could potentially make a difference is a good strategy, particularly when you're in the first few rounds of annotations. So, for example, making sure that all of your texts come from the same time period, or checking that all of your speakers are native to the language you are asking them to speak in are things that you may want to take into account even if you ultimately decide to not include that type of diversity in your corpus.

Summary

- Having a clear definition of your annotation task will help you stay on track when you start creating task definitions and writing annotation guidelines
- There is often a trade-off in annotation tasks between informativity and accuracy —be careful that you aren't sacrificing too much of one in favor of another.
- Clearly defining the scope of your task will make decisions about the sources of your corpus—and later, the annotation tags and guidelines—much easier.
- Doing some background research can help keep you from reinventing the wheel when it comes to your own annotation task.
- Using an existing corpus for your dataset can make it easier to do other analyses if necessary.
- If an existing corpus doesn't suit your needs then you can build your own, but consider carefully what data you need and what might become a confounding factor.
- There are a variety of existing tools and programming languages that can help you to collect data from the Internet.
- What information you intend to show to your annotators is an important factor that can influence annotation, particularly in tasks that rely on opinion or annotator's interpretations of texts, rather than objective facts.

CHAPTER 3

Building Your Model and Specification

Now that you've defined your goal and collected a relevant dataset, you need to create the model for your task. But what do we mean by "model"? Basically, the model is the practical representation of your goal: a description of your task that defines the classifications and terms that are relevant to your project. You can also think of it as the aspects of your task that you want to capture within your dataset. These classifications can be represented by metadata, labels that are applied to the text of your corpus, and/or relationships between labels or metadata. In this chapter, we will address the following questions:

- The model is captured by a specification, or spec. But what does a spec look like?
- I've got the goals for my annotation project. Where do I start? How do turn a goal into a model?
- What form should my model take? Are there standardized ways to structure the phenomena?
- How do you take someone else's standard and use it to create a specification?
- What do you do if there are no existing specifications, definitions, or standards for the kinds of phenomena you are trying to identify and model?
- How do you determine when a feature in your description is an element in the spec versus an attribute on an element?

The spec is the concrete representation of your model. So while the model is an abstract idea of what information you want your annotation to capture, and the interpretation of that information, the spec turns those abstract ideas into tags and attributes that will be applied to your corpus.

Some Example Models and Specs

Recall from Chapter 1, that the first part in the MATTER cycle involves creating a model for the task at hand. We introduced a model as a triple, $M = <T,R,I>$, consisting of a vocabulary of terms, T, the relations between these terms, R, and their interpretation,

I. However, this is a pretty high-level description of what a Model is. So before we discuss more theoretical aspects of models, let's look at some examples of annotation tasks, and see what the models for those look like.

For the most part, we'll be using XML DTD (Document Type Definition) representations. XML is becoming the standard for representing annotation data, and DTDs are the simplest way to show an overview of the type of information that will be marked up in a document. The next few sections will go through what the DTDs for different models will look like, so you can see how the different elements of an annotation task can be translated into XML-compliant forms.

What is a DTD?

A DTD is a set of declaractions containing the basic building blocks that allow an XML document to be validated. DTDs have been covered in depth in other books and websites (Learning XML, XML in a Nutshell, W3schools), but we'll give an short overview here.

Essentially, the DTD defines what the structure of an XML document will be by defining what tags will be used inside the document, and what attributes those tags will have. By having a DTD, the XML in a file can be validated to ensure that the formatting is correct.

So what do we mean by tags and attributes? Let's take a really basic example: web pages and HTML. If you've ever made a website and edited some code by hand, you're familiar with elements such as <b> and <br />. These are tags that tell a program reading the HTML that the text in between <b> and </b> should be **bold** and that <br /> indicates a newline should be included when the text is displayed. Annotation tasks use similar formatting, but they define their own tags based on what information is considered important for the goal being pursued. So an annotation task that is based on marking the parts of speech in a text might have tags such as <noun>, <verb>, <adj>, and so on. In a DTD, these tags would be defined like this:

```
<!ELEMENT noun ( #PCDATA ) >
<!ELEMENT verb ( #PCDATA ) >
<!ELEMENT adj ( #PCDATA ) >
```

The string !ELEMENT indicates that the information contained between the < and > is about an element (also known as a "tag"), and the word following it is the name of that tag (noun, verb, adj). The (#PCDATA) indicates that the information between the <noun> and </noun> tags will be parsable character data (other flags instead of #PCDATA can be used to provide other information about a tag, but for this book we're not going to worry about them).

By declaring the three tags above in a DTD, we can now have a valid XML document that has nouns, verbs, and adjectives all marked up. However, annotation tasks often require more information about a piece of text than just its type. This is where *attributes* come in. For example, knowing that a word is a verb is useful, but it's even more useful to know the tense of the verb—past, present, or future. This can be done by adding an attribute to a tag, which looks like this:

```
<!ELEMENT verb ( #PCDATA ) >
<!ATTLIST verb tense ( past | present | future | none ) #IMPLIED >
```

The !ATTLIST line declares that an attribute called "tense" is being added to the verb element, and that it has four possible values: past, present, future, and none. The #IMPLIED shows that the information in the attribute isn't required for the XML to be valid (again, don't worry too much about this for now). Now you can have a verb tag that looks like this:

```
<verb tense="present">
```

You can also create attributes that allow annotators to put in their own information, by declaring the attribute's type to be CDATA insteas of a list of options, like this:

```
<!ELEMENT verb ( #PCDATA ) >
<!ATTLIST verb tense CDATA #IMPLIED >
```

One last type of element that is commonly used in annotation is a *linking element*, or a *link tag*. These tags are used to show relationships between other parts of the data that have been marked up with tags. For instance, if the part-of-speech task also wanted to show the relationship between a verb and the noun that performed the action described by the verb, the annotation model might include a link tag called "performs", like so:

```
<!ELEMENT performs EMPTY >
<!ATTLIST performs fromID IDREF >
<!ATTLIST performs toID IDREF >
```

The EMPTY in this element tag indicates that the tag will not be applied to any of the text itself, but rather is being used to provide other information about the text. Normally in HTML an empty tag would be something like the <br /> tag, or another tag that stands one its own. In annotation tasks, an empty tag is used to create paths between other, contentful tags.

In a model, it is almost always important to keep track of the order (or *arity*) of the elements involved in the linking relationship. We do this here by using two elements that have the type IDREF, meaning that they will refer to other annotated extents or elements in the text by identifiable elements.

We'll talk more about the IDs and the relationship between DTDs and annotated data in Chapter 4, but for now this should give you enough information to understand the examples provided in this chapter.

There are other formats that can be used to specify specs for a model. XML schema are sometimes used to create a more complex representation of the tags being used, as is the Backus–Naur Form. However, these formats are more complex than DTDs, and aren't generally necessary to use unless you are using a particular piece of annotation software, or want to have a more restrictive spec. For the sake of simplicity, we will use only DTD examples in this book.

Film genre classification

A common task in natural language processing and machine learning is classifying documents into categories; for example, using film reviews or summaries to determine the genre of the film being described. If you have a goal of being able to use machine learning to identify the genre of a movie from the movie summary or review, then a corresponding model could be that you want to label the summary with the all the genres that the movie applies to, in order to feed those labels into a classifier and train it to identify relevant parts of the document. In order to turn that model into a spec, you need to think about what that sort of label would look like, presumably in a DTD format.

The easiest way to create a spec for a classification task is to simply create a tag that captures the information that you need for your goal and model. In this case, you could create a tag called “genre” that has an attribute called “label”, where “label” holds the values that can be assigned to the movie summary. The simplest incarnation of this spec would be this:

```
<!ELEMENT genre ( #PCDATA ) >
<!ATTLIST genre label CDATA #IMPLIED >
```

This DTD has the required tag and attribute, and allows for any information to be added to the “label” attribute. Functionally for annotation purposes, this means that the annotator would be responsible for filling in the genres that he or she thinks apply to the text. Of course, there are a large number of genre terms that have been used, and not everyone will agree on what a “standard” list of genres should be—for example, are “fantasy” and “sci-fi” different genres, or should they be grouped into the same category? Are “mystery” films different from “noir”? Because the list of genres will vary from person to person, it might be better if your DTD specified a list of genres that annotators could choose from, like this:

```
<!ELEMENT genre ( #PCDATA ) >
<!ATTLIST genre label ( Action | Adventure | Animation | Biography | Comedy | Crime |
    Documentary | Drama | Family | Fantasy | Film-Noir | Game-Show | History |
    Horror | Music | Musical | Mystery | News | Reality-TV | Romance |
    Sci-Fi | Sport | Talk-Show | Thriller | War | Western ) >
```

The list in the “label” attribute above is taken from the IMDB’s list of genres. Naturally, since other genre lists exist, you would want to choose the one that best matches your task, or create your own list. As you go through the process of annotation and the rest of the MATTER cycle, you’ll find places where your model/spec needs to be revised in order to get the results that you want. This is perfectly normal, even for tasks that seem as straightforward as putting genre labels on movie summaries—annotator opinions can vary, even when the task is as clearly defined as you can make it, and computer algorithms don’t really think and interpret the way that people do, so even when you get past the annotation phase, you may still find places where, in order to maximize the correctness of the algorithm you would have to change your model.

For example, looking at the genre list from IMDB we see that “romance” and “comedy” are two separate genres, and so the summary of a romantic comedy would have to have two labels: romance and comedy. But if in a significant portion of reviews those two tags appear together, an algorithm may learn to *always* associate the two, even when the summary being classified is really a romantic drama or musical comedy. So, you might find it necessary to create a “rom-com” label in order to keep your classifier from creating false associations.

In the other direction, there are many historical action movies that take place over very different periods in history, and a machine learning algorithm may have trouble finding enough common ground between a summary of 300, Braveheart, and Pearl Harbor to create an accurate associate with the “history” genre. In that case, you might find it necessary to add different levels of historical genres, ones that reflect different periods in history in order to train a classifier in the most accurate way possible.

If you’re unclear on how the different components of the machine learning algorithm can be affected by the spec, or why you might need to adapt a model to get better results, don’t worry! For now, just focus on turning your goal into a set of tags, and the rest will come later. But if you really want to know how all this works, Chapter 6 has an overview of all the different ways that machine learning algorithms “learn”, and what it means to train each one.

Adding Named Entities

Of course, reworking the list of genres isn’t the only way to change a model to better fit a task. Another way is to add tags and attributes that will more closely reflect the information that‘s relevant to your goal. In the case of the movie summaries, it might be useful to keep track of some of the named entities that appear in the summaries that may give insight into the genre of the film. A named entity is an entity (an object in the world) that has a name which uniquely identifies it by name, nickname, abbreviation, etc. "O'Reilly", “Brandeis University”, “Mount Hood”, “IBM”, and “Vice President” are all examples of named entities. In the movie genre task, it might be helpful to keep track of named entities such as film titles, directors, writers, actors, and characters that are mentioned in the summaries.

You can see from the above list that there are many different named entities (NEs) in the model that we would like to capture. Because the model is abstract, the practical application of these NEs to a spec or DTD has to be decided upon. There are often many ways in which a model can be represented in a DTD, due to the categorical nature of annotation tasks and of XML itself. In this case there are two primary ways in which the spec could be created: a single tag called “named entity” with an attribute that would have each of the items from the above list, like this:

```
<!ELEMENT named_entity ( #PCDATA ) >
<!ATTLIST named_entity role (film_title | director | writer | actor | character ) >
```

or, each role could be given its own tag, like this:

```
<!ELEMENT film_title ( #PCDATA ) >
<!ELEMENT director ( #PCDATA ) >
<!ELEMENT writer ( #PCDATA ) >
<!ELEMENT actor ( #PCDATA ) >
<!ELEMENT character ( #PCDATA ) >
```

While these two specs seem to be very different, in many ways they are interchangeable. It would not be difficult to take an XML file with the first DTD and change it to one that is compliant with the second. Often the choices that you'll make about how your spec will represent your model will be influenced by other factors, such as what format is easier for your annotators, or what works better with the annotation software you are using. We'll talk more about the considerations that go into which formats to use in Chapter 4 and (to come).

By giving machine learning algorithms more information about the words in the document that are being classified, such as by annotating the named entities, it's possible to create more accurate representations of what's going on in the text, and to help the classifier pick out markers that might make the classifications better.

Semantic Roles

Another layer of information that might be useful in examining movie summaries is to annotate the relationships between the named entities that are marked up in the text. These relationships are called *semantic roles*, and they are used to explicitly show the connections between the elements in a sentence. In this case, it could be helpful to annotate the relationships between actors and characters, and the staff of the movie and which movie they worked on. Consider the following example summary/review:

"In *Love, Actually*, writer/director Richard Curtis weaves a convoluted tale about characters and their relationships. Of particular note is Liam Neeson (Schindler's List, Star Wars) as Daniel, a man struggling to deal with the death of his wife and the relationship with his young step-son, Sam (Thomas Sangster). Emma Thompson (Sense and Sensibility, Henry V) shines as a middle-aged housewife whose marriage with her husband (played by Alan Rickman) is under siege by a beautiful secretary. While this movie does have its purely comedic moments (primarily presented by Bill Nighy as out-of-date rock star Billy Mack), this movie avoids the more in-your-face comedy that Curtis has presented before as a writer for Black Adder and Mr. Bean, presenting instead a remarkable, gently humorous insight into what love, actually, is."

Using one of the named entity DTDs from the previous section would lead to a number of annotated extents, but due to the density an algorithm may have difficulty determining who goes with what. By adding semantic role labels such as acts_in, acts_as, directs, writes, and character_in, the relationships between all the named entities will become much clearer.

As with the DTD for the named entities, we are faced with a choice between using a single tag with multiple attribute options:

```
<!ELEMENT sem_role ( EMPTY ) >
<!ATTLIST sem_role from IDREF >
<!ATTLIST sem_role to IDREF >
<!ATTLIST sem_role label (acts_in | acts_as | directs | writes | character_in ) >
```

or a tag for each semantic role we wish to capture:

```
<!ELEMENT acts_in ( EMPTY ) >
<!ATTLIST acts_in from IDREF >
<!ATTLIST acts_in to IDREF >

<!ELEMENT acts_as ( EMPTY ) >
<!ATTLIST acts_as from IDREF >
<!ATTLIST acts_as to IDREF >

<!ELEMENT directs ( EMPTY ) >
<!ATTLIST directs from IDREF >
<!ATTLIST directs to IDREF >

<!ELEMENT writes ( EMPTY ) >
<!ATTLIST writes from IDREF >
<!ATTLIST writes to IDREF >

<!ELEMENT character_in ( EMPTY ) >
<!ATTLIST character_in from IDREF >
<!ATTLIST character_in to IDREF >
```

you'll notice that this time the DTD specifies that each of these elements is EMPTY, meaning that no character data is associated directly with the tag. Remember that linking tags in annotation are usually defined by EMPTY tags specifically because links between elements do not generally have text associated with them specifically, but rather clarify a relationship between two or more other extents. We'll discuss more about the application of linking and other types of tags in Chapter 4.

Adopting (or not Adopting) Existing Models

Now that you have an idea of how spec can represent a model, let's look a little more closely at some of the details we just presented you with. You might recall from Chapter 1 that when we discussed semantic roles we presented a very different list from "acts_in, acts_as, directs, writes, and character_in". Here's what the list looked like:

- agent: The event participant that is doing or causing the event to occur;
- theme/figure: The event participant who undergoes a change in position or state;
- experiencer: The event participant who experiences or perceives something;
- source: The location or place from which motion begins; The person from whom the theme is given;

- goal: The location or place to which the motion is directed or terminates;
- recipient: The person who comes into possession of the theme;
- patient: The event participant who is affected by the event;
- instrument: The event participant used by the Agent to do or cause the event;
- location/ground: The location or place associated with the event itself.

Similarly, we also presented an ontology that defined the categories Organization, Person, Place, and Time. This set of labels can be viewed as a simple model of named entity types that are commonly used in other annotation tasks.

So if these models, exist, why didn't we just use them for our film genre annotation task? Why did we create our own sets of labels for our spec? Just as when defining the goal of your annotation you need to think about the trade-off between informativity and correctness, when creating the model and spec for your annotation task you need to consider the trade off between *generality* and *specificity*.

Creating your own Model and Specification: Generality versus Specificity

The ontology consisting of Organization, Person, Place, and Time is clearly a very general model for entities in a text, but for the film genre annotation task it is much too general to be useful for the kinds of distinctions we want to be able to make. Of the named entities labels that we identified earlier, 4 of them (director, writer, actor, and character) would all fall under the label "Person", and "film title" doesn't clearly fit under any of them. Using these labels would lead to unhelpful annotations in two respects: first, the labels used would be so generic as to be useless for the task (labeling everyone as "person" won't help distinguish one movie review from another); and secondly, it would be difficult to explain to the annotators that, while you've given them a set of labels, you don't want every instance of those types of entities labeled, rather only the ones that are relevant to the film (so, for example, a mention of another reviewer would not be labeled as a "person"). Clearly, overly general tags in a spec can lead to confusion during annotation.

On the other hand, we could have made the tags in the spec even more specific, such as: actor_star, actor_minor_character, character_main, character_minor, writer_film, writer_book, writer_book_and_film, and so on. But what would be gained from such a complicated spec? While it's possible to think of an annotation task where it might be necessary to label all that information (perhaps one that was looking at how these different people are described in movie reviews), remember that the task we defined was first simply labeling the genres of films as they are described in summaries and reviews, and was then expanded to include some other information that might be relevant to making that determination. Using overly specific tags in this case would decrease how useful the annotations would be, and also increase the amount of work done by the annotators for no obvious benefit. Figure 3-1 shows the different levels of

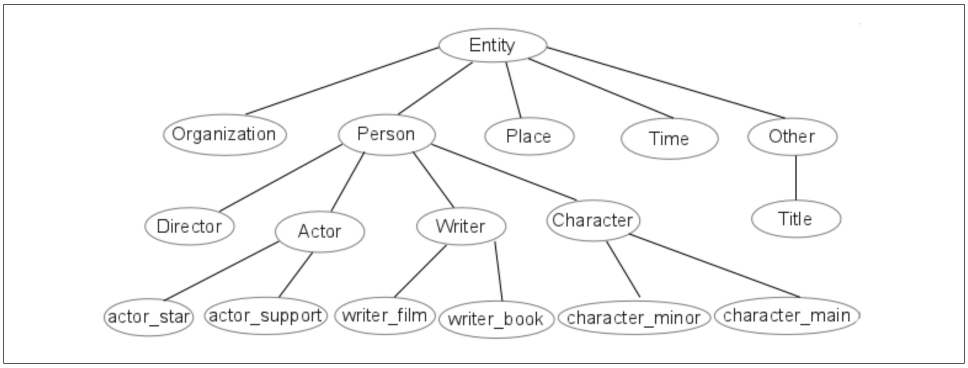


Figure 3-1. A heirarchy of named entities.

the heirarchy we are discussing. The top two levels are too vague, while the bottom is too specific to be useful. The third level is just right for this task.

We face the same dichotomy when examining the list of semantic roles. The list given in linguistic textbooks is a very general list of roles that can be applied to the nouns in a sentence, but any annotation task trying to use them for film-related roles would have to have a way to limit which nouns were assigned roles by the annotator, and most of the roles related to the named entities we're interested in would simply be "agent"—a label that is neither helpful nor interesting for this task. So in order to create a task that was in the right place regarding generality and specificity, we developed our own list of roles that were particular to this task.

We haven't really gotten into the details of named entity and semantic role annotation using existing models, but these are not trivial annotation tasks. If you're interested in learning more about annotation efforts that use these models, check out FrameNet for semantic roles, and the Message Understanding Conferences (MUCs) for examples of named entity and coreference annotation.

Overall, there are a few things that you want to make sure your model and specification have in order to proceed with your task. They should:

- contain a representation of all the tags and links relevant to completing your goal.
- be relevant to the implementation stated in your goal (If your purpose is to classify documents by genre, spending a lot of time annotating temporal information is probably not going to be of immediage help).
- be grounded in existing research as much as possible. Even if there's no existing annotation spec that meets your goal completely, you can still take advantage of research that's been done on related topics, which will make your own research much easier.

Specifically to the last point on the list, even though the specs we've described for the film genre annotation task use sets of tags that we created for this purpose, it's difficult to say that they weren't based on an existing model to some extent. Obviously some knowledge about named entities and semantic roles helped to inform how we described the annotation task, and helped us to decide whether annotating those parts of the document would be useful. But you don't need to be a linguist to know that nouns can be assigned to different groups, and that the relationships between different nouns and verbs can be important to keep track of. Ultimately, while it's entirely possible that your annotation task is completely innovative and new, it's still worth taking a look at some related research and resources and seeing if any of them are helpful for getting your model and spec put together.

The best way to find out if a spec exists for your task is to do a search for existing annotated datasets

Using Existing Models and Specifications

While the examples we discussed above had fairly clear-cut reasons for us to create our own tags for the spec, there are some advantages to basing your annotation task on existing models. Interoperability is a big concern in the computer world, and it's actually a pretty big concern in linguistics as well—if you have an annotation that you want to share with other people, there are a few things that make it easier to share, such as using existing annotation standards (such as standardized formats for your annotation files), using software to create the annotation that other people can also use, making your annotation guidelines available to other people, and using models or specifications that have been already vetted in similar tasks. We'll talk more about standards and formats later in this chapter and in the next one; for now we'll focus just on models and specs.

By using models or specs that other people have used, there are a few ways that your project can benefit. First of all, if you use the specification from an annotation project that's already been done, you have the advantage of using a system that's already been vetted, and one that may also come with an annotated corpus, which you can use to train your own algorithms or use to augment your own dataset (assuming that the usage restrictions on the corpus allow for that, of course).

In "Background research" on page 38 we mentioned some places to start looking for information that would be useful with defining your goal, so presumably you've already done some research into the topics you're interested in (if you haven't now is a good time to go back and do so). Even if there's no existing spec for your topic, you might find a descriptive model like the one we provided for semantic roles.

Not all annotation and linguistic models live in semantic textbooks! The list of film genres that we used was taken from imdb.com, and there are many other places where you can get insight into how to frame your model and specification. A recent paper on annotating bias used the Wikipedia standards for editing pages as the standard for developing a spec and annotation guidelines for an annotation project (Herzig et al., 2011). Having a solid linguistic basis for your task can certainly help, but don't limit yourself to only linguistic resources!

If you are lucky enough to find both a model and a specification that is suitable for your task, you still might need to make some changes in order for it to fit your goal. For example, if you are doing temporal annotation you can start with the TimeML specification, but you may find that the Timex3 tag is simple too much information for your purposes, or too overwhelming for your annotators. The Timex3 DTD description is here:

```
<!ELEMENT TIMEX3 ( #PCDATA ) >
<!ATTLIST TIMEX3 start #IMPLIED >
<!ATTLIST TIMEX3 tid ID #REQUIRED >
<!ATTLIST TIMEX3 type ( DATE | DURATION | SET | TIME ) #REQUIRED >
<!ATTLIST TIMEX3 value NMTOKEN #REQUIRED >
<!ATTLIST TIMEX3 anchorTimeID IDREF #IMPLIED >
<!ATTLIST TIMEX3 beginPoint IDREF #IMPLIED >
<!ATTLIST TIMEX3 endPoint IDREF #IMPLIED >
<!ATTLIST TIMEX3 freq NMTOKEN #IMPLIED >
<!ATTLIST TIMEX3 functionInDocument ( CREATION_TIME | EXPIRATION_TIME | MODIFICATION_TIME |
     PUBLICATION_TIME | RELEASE_TIME | RECEPTION_TIME | NONE ) #IMPLIED >
<!ATTLIST TIMEX3 mod ( BEFORE | AFTER | ON_OR_BEFORE | ON_OR_AFTER | LESS_THAN | MORE_THAN |
     EQUAL_OR_LESS | EQUAL_OR_MORE | START | MID | END | APPROX )  #IMPLIED >
<!ATTLIST TIMEX3 quant CDATA #IMPLIED >
<!ATTLIST TIMEX3 temporalFunction ( false | true ) #IMPLIED >
<!ATTLIST TIMEX3 valueFromFunction IDREF #IMPLIED >
<!ATTLIST TIMEX3 comment CDATA #IMPLIED >
```

There is a lot of information being encoded in a Timex3 tag. While the information is there for a reason—years of debating and modifying took place in order to create this description of a temporal reference—there are certainly annotation tasks where this level of detail will be unhelpful, or even detrimental. If this is the case, however, there are other temporal annotation tasks that have been done over the years that have specs that you may find more suitable for your goal and model.

Using Models without Specifications

It's entirely possible—even likely—that your annotation task may be based on a linguistic (or psychological or sociological) phenomena that has been clearly explained in the literature, but has not yet been turned into a specification. In that case, you will be have to make decisions about the form the specification will take, in much the same way that we discussed in the first section of this chapter. Depending on how fleshed out the model is, you may have to make decisions about what parts of the model become tags, what become attributes, and what become links. In some ways this can be harder than simply creating your own model and spec, because you will be somewhat constrained by someone else's description of the phenomena. However, having a specification that is grounded in an established theory will make your own work easier to explain and distribute, so there are advantages to this approach as well.

Many (if not all) of the annotation specifications that are currently in wide use are based on theories of language that were created prior to the annotation task being created. For example, the TLINK tag in ISO-TimeML is based largely on James Allen's work in temporal reasoning (Allen, 1984; Pustejovsky et al., 2003), and ISO-Space has been influenced by the qualitative spatial reasoning work of Randell et al. (1992) and others. Similarly, syntactic bracketing and part-of-speech labeling work, as well as existing semantic role labeling are all based on models developed over years of linguistic research and then applied through the creation of syntactic specifications.

Different Kinds of Standards

Previously we mentioned that one of the aspects of having an interoperable annotation project is using a standardized format for your annotation files, as well as using existing models and specs. However, file specifications are not the only kind of standards that exist in annotation: there are also annotation specifications that have been accepted by the community as go-to (or *de facto*) standards for certain tasks. While there are no mandated (aka *de jure*) standards in the annotation community, there are varying levels and types of de facto standards that we will discuss here.

ISO Standards

The International Organization for Standardization (ISO) is the body responsible for creating standards that are used around the world for ensuring compatibility of systems between businesses and government, and across borders. The ISO is the organization that helps determine what the consensus will be for many different aspects of daily life, such as the size of DVDs, representation of dates and times, and so on. There are even ISO standards for representing linguistic annotations in general and for certain types of specifications, in particular ISO-TimeML and ISO-Space. Of course, it isn't *required* to use ISO standards (there's no Annotation Committee that enforces use of these standards), but they do represent a good starting point for most annotation tasks, particularly those standards related to representation.

ISO standards are created with the intent of interoperability, which sets them apart from other de facto standards, as those often become the go-to representation simply because they were there first, or were used by a large community at the outset and gradually became ingrained in the literature. While this doesn't mean that non-ISO standards are inherently problematic, it does mean that they may not have been created with interoperability in mind.

Annotation format standards

Linguistic annotation projects are being done all over the world for many different, but often complementary reasons. Because of this, in the past few years the International Organization for Standardization (ISO) has been developing the Linguistic Annotation Framework (LAF), a model for annotation projects that is abstract enough to apply to any level of linguistic annotation.

How can a model be flexible enough to encompass all different types of annotation tasks? LAF takes a two-pronged approach to standardization. First, it focuses on the structure of the data, rather than the content. Specifically, the LAF standard allows for annotations to be represented in any format that the task organizers like, so long as it can be transmuted into LAF's XML-based "dump format", which acts as an interface for all manners of annotations. The dump format has the following qualities (Ide and Romary, 2006):

- Annotation is kept separate from the text it is based on, and annotations are associated with character or element offsets derived from the text.
- Each level of annotation is stored in a separate document
- Annotations that represent hierarchical information (for example, syntax trees) must be either represented with embedding in the XML dump format, or using a flat structure that symbolically represents relationships
- When different annotations are merged, the dump format must be able to integrate overlapping annotations in a way that is compatible with XML

The first bullet point—keeping annotation separate from the text—now usually takes the form of stand-off annotation (as opposed to inline annotation, where the tags and text are intermingled). We'll go through all the forms that annotation can take and the pros and cons in Chapter 4.

The other side of the approach that LAF takes towards standardization is encouraging researchers to use established labels for linguistic annotation. What this means is that instead of just creating your own set of part-of-speech or named entity tags, you can go to the Data Category Registry (DCR) for definitions of existing tags, and use those to model your own annotation task. Alternatively, you can name your tag whatever you want, but when transmuting to the dump format you would provide information about what tags in the DCR your own tags are equivalent to. This will help other people merge existing annotations, because it will be known whether two annotations are equivalent despite naming differences. The DCR is currently under development (it's not an easy task to create a repository of all annotation tags and levels, and so progress has been made very carefully). You can see the information as it currently exists at ISOcat.

Timeline of Standardization

LAF didn't emerge as an ISO standard from out of nowhere. Here's a quick rundown of where these standards originated:

- **1987** - Text Encoding Initiative (TEI) founded "to develop guidelines for encoding machine-readable texts in the humanities and social sciences". The TEI is still an active organization today. *www.tei-c.org*
- **1990** - The TEI released its first set of *Guidelines for the Encoding and Interchange of Machine Readable Texts*. It recommends that encoding be done using SGML (Standard Generalized Markup Language), the precursor to XML and HTML.
- **1993** - The Expert Advisory Group on Language Engineering Standards (EAGLES) formed to provide standards for large-scale language resources (ex, corpora), as well as standards for manipulating and evaluating those resources. *http://www.ilc.cnr.it/EAGLES/home.html*
- **1998** - The Corpus Encoding Standard (CES), also based on SGML, is released. The CES is a corpus-specific application of the standards laid out in the TEI's *Guidelines* and was developed by the EAGLES group. *http://www.cs.vassar.edu/CES/*
- **2000** - The Corpus Encoding Standard for XML (XCES) was released, again under the EAGLES group. *http://www.xces.org/*
- **2002** - The TEI released version p4 of their *Guidelines*, the first version to implement XML. *http://www.tei-c.org/Guidelines/P4/*
- **2004** - The first document describing the Linguistic Annotation Framework is released (Ide and Romary, 2004)
- **2007** - The most recent version (p5) of the TEI is released. *http://www.tei-c.org/Guidelines/P5/*
- **2012** - LAF and the TEI Guidelines are still being updated and improved to reflect progress made in corpus and computational linguistics

Annotation specification standards

In addition to helping create standards for annotation formats, the ISO is also working on developing standards for specific annotation tasks. We've mentioned ISO-TimeML already, which is the standard for representing temporal information in a document. There is also ISO-Space, the standard for representing locations, spatial configurations, and movement in natural language. The area of ISO that is charged with looking at annotation standards for all areas of natural language is called TC 37/SC 4. Other projects involve the development of standards for: how to encode syntactic categories and morphological information in different languages; semantic role labeling; dialogue act labeling; discourse relation annotation; and many others. For more information, you can visit the ISO webpage or check out the appendix on corpora and specification languages.

Community-driven standards

In addition to the committee-based standards provided by the ISO, there are a number of de facto standards that have developed in the annotation community simply through wide use. These standards are created when an annotated resource is created and made available for general use. Because corpora and related resources can be very time-consuming to create, once a corpus is made available it will usually quickly become part of the literature. By extension, whatever annotation scheme was used for that corpus will also tend to become a standard.

If there is a spec that is relevant to your project, taking advantage of community-driven standards can provide some very useful benefits for your annotation project. Any existing corpora that are related to your effort will be relevant, since they are developed using the spec you want to adopt. Additionally, because resources like these are often in wide use, searching the literature for mentions of the corpus will often lead you to papers that are relevant to your own research goals, and will help you identify any problems that might be associated with the dataset or specification. Finally, for datasets that have been around long enough, there are often tools and interfaces that have been built around them that will make them easier for you to use.

Community-driven standards don't necessarily follow LAF guidelines, or make use of other ISO standards. This doesn't mean that they should be disregarded, but if interoperability is important to you, you may have to do a little extra work to make your corpus fit the LAF guidelines.

We have a list of existing corpora in Appendix C to help you get started finding resources that are related to your own annotation task. While the list is as complete as we could make it, it is not exhaustive and you should still check online for resources that would be useful to you. The list that we have was compiled from the LRE-map, a database of natural language processing resources maintained by the European Language Resource Association (ELRA).

Other standards affecting annotation

While the ISO and community-driven standards are generally the only standards *directly* related to annotation and natural language processing, there are many standards in day-to-day life that can affect your annotation project. For example, the format that you choose to store your data in (unicode, UTF-8, UTF-16, ASCII, etc.) will affect how easily other people will be able to use your texts on their own computers. This becomes especially tricky if you are annotating in a language other than English, where the alphabet uses different sets of characters. Even languages with characters that overlap with English (French, Spanish, Italian, etc) can be problematic when accented vowels are used. We recommend using UTF-8 for encoding most languages, as it is an encoding that captures most characters that you will encounter, and it is available for nearly all computing platforms.

Other standard that can affect a project are those that vary by region, such as the representation of dates and times. If you have a project where it is relevant to know when the document was created, or how to interpret the dates in the text, it's often necessary to know where the document originated. In the United States, dates are often represented as MM-DD-YYYY, whereas in other countries the dates are written DD-MM-YYYY. So if you see the date 01-03-1999 in a text, knowing where it's from might help determine whether the date is January 3rd or March 1st. Adding to the confusion, most computers will store dates as YYYY-MM-DD so they can be easily sorted.

Similarly, name conventions can also cause confusion. When annotating named entities, if you're making a distinction between given names and family names, again, the origin of the text can be a factor in how the names should be annotated. This can be especially confusing, because while it might be a convention in a country for people to be referred to by their family name first (as in Hungary, South Korea, or Japan), if the text you are annotating has been translated the names may have been (or may not have been) swapped by the translator to follow the convention of the language being translated to.

None of the issues we've mentioned should be deal breakers for your project, but they are definitely things to be aware of. Depending on your task, you may also run into regional variations in language or pronunciation, which can be factors that you should take into account when creating your corpus. Additionally, you may need to modify your model or specification to allow for annotating different formats of things like dates and names if you find that your corpus has more diversity in it than you initially thought.

Summary

- The model of your annotation project is the abstract representation of your goal, and the specification is the concrete representation of it.
- XML DTDs are a handy way to represent a specification; they can be applied directly to an annotation task
- Most models and specifications can be represented using three types of tags: document-level labels, extent tags, and link tags.

- When creating your specification, you will need to consider the tradeoff between generality and specificity. Going too far in either direction can make your task confusing and unmanagable.
- Searching existing datasets, annotation guidelines and related publications and conferences is a good way to find existing models and specifications for your task.
- Even if no existing task is a perfect fit for your goal, modifying an existing specification can be a good way to keep your project grounded in linguistic theories.
- Interoperability and standardization are concerns if you want to be able to share your projects with other people. In particular, text encoding and annotation format can have a big impact on how easily other people can use your corpus and annotations.
- Both ISO standards and community-driven standards are useful bases for creating your model and specification.
- Regional differences in standards of writing, text representation and other natural language conventions can have an effect on your task, and may need to be represented in your specification.

CHAPTER 4

Applying and Adopting Annotation Standards to your Model

Now that you've created the spec for your annotation goal, you're getting ready to actually start annotating your corpus. However, before you get to annotating you need to consider what form your annotated data will take—that is to say, you know **what** you want your annotators to do, but you have to decide **how** you want them to do it. In this chapter we'll examine the different formats annotation can take, and discuss the pros and cons of each one by looking at the following questions:

- What does annotation look like?
- Are different types of tasks represented differently? If so, how?
- How can you ensure your annotation can be used by other people and in conjunction with other tasks?
- What considerations go into deciding on an annotation environment and data format, both for the annotators and machine learning?

Annotated corpora

Before getting into the details of how to apply your spec to your corpus, you need to understand what annotation actually looks like when it's been applied to a document or text. So now lets look at the spec examples from Chapter 3 and see how they can be applied to an actual corpus.

There are many different ways to represent information about a corpus. The examples we show you won't be exhaustive, but they will give you an overview of some of the different formats that annotated data can take.

Keep your data accessible. Your annotation project will be much easier to manage if you choose a format for your data that's easy for you to modify and access. Using intricate database systems or complicated XML schemas to define your data is fine if you're used to them, but if you aren't then you'll be better off keeping things simple.

Annotation tasks range from simple document labeling to text extent tagging and tag linking. As specs and tasks grow in complexity, more information is needed to be contained within the annotation. In the following sections we'll discuss the most common ways that these tasks are represented in data, and the pros and cons of each style.

Metadata annotation: Document classification

In Chapter 3 we discussed one examples of document classification tasks, that of labeling the genres of a movie based on a summary or review by using non-exclusive category labels (a movie can be both a comedy and a Western, for example). However, before we get to multiple category labels for a document, let's look at a slightly simpler example: labeling movie reviews as positive, negative, or neutral towards the movie they are reviewing. This is a simpler categorization exercise because the labels will not overlap; each document will have only a single classification.

Unique labels - movie reviews

Let's say that you have a corpus of 100 movie reviews, with roughly equal amounts of positive, negative and neutral documents. By reading each document you (or your annotators) can determine which category each document should be labeled as, but how are you going to represent that information? Here are a few suggestions for what you can do:

- Have a text file or other simple file format (comma-separated, for example) containing a list of file names, and its associated label
- Create a database file and have your annotators enter SQL commands to add files to the appropriate table
- Create a folder for each label on your computer, and as each review is classified, move the file into the appropriate folder
- Have the annotators change the filename to add "positive", "negative" or "neutral", as in "review0056-positive.txt"
- Add the classification inside the file containing the review

Notice that these options run the gamut from completely external representations of the annotation data, where the information is stored in completely different files, to entirely internal, where the information is kept inside the same document. They also cover the middle ground, where the information is kept in the file system; near the corpus but not completely part of it.

So which of these systems is best? In Chapter 3, we discussed the importance of the LAF and GrAF standards, and explained why standoff annotation is preferable to other making changes to the actual text of the corpus. So the last option on the list isn't one that's preferable.

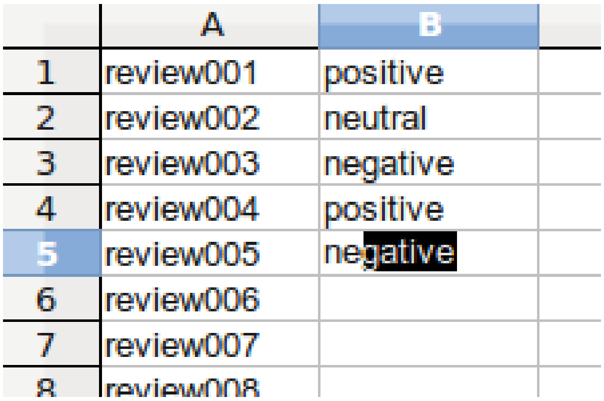
But how do you choose between the other four options? They are all recording the annotation information while still preserving the format of the data; is one really better than the other? In terms of applying the model to the data, we would argue that no, there's no real difference between any of the remaining options. Each representation could be turned into any of the others without loss of data or too much effort (assuming that you or someone you know can do some basic programming, or is willing to do some reformatting by hand).

So the actual decision here is going to be based on other factors, such as what will be easiest for your annotators and what will result in the most accurate annotations. Asking your annotators to learn SQL commands in order to create tables might be the best option from your perspective, but unless your annotators are already familiar with that language and an accompanying interface, chances are that using such a system will greatly slow the annotation process, and possibly result in inaccurate annotations or even loss of data if someone manages to delete your database.

Be aware of sources of error! Annotation tasks are often labor-intensive and require attention to detail, so any source of confusion or mistakes will probably crop up at least a few times.

Having your annotators type information can also be problematic, even with a simple labeling task like this one. Consider giving your annotators a folder of text files and a spreadsheet containing a list of all the files names. If you ask your annotators to fill in the spreadsheet slots next to each file name with the label, what if they are using a program that will "helpfully" suggest options for auto filling each box? If you are using the labels "positive", "negative" and "neutral", then the last two both start with "ne", and if an annotator gets tired or doesn't pay attention, they may find themselves accidentally filling in the wrong label. In a situation like that, you might want to consider using a different set of words, such as "likes", "dislikes" and "indifferent".

Of course, this doesn't mean that it's impossible to complete a task by using a spreadsheet and classifications that are a bit similar. In some cases, such circumstances are impossible to avoid. However, it's never a bad idea to keep an eye out for places where mistakes can easily slip in.



	A	B
1	review001	positive
2	review002	neutral
3	review003	negative
4	review004	positive
5	review005	negative
6	review006	
7	review007	
8	review008	

Figure 4-1. A possible source of error in annotation

While we didn't discuss the movie review annotation scenario in Chapter 3, we have assumed here that we have a schema that contains three categories. However, that is by no means the only way to frame this task and to categorize movie reviews. In the Movie Review Corpus that comes with the NLTK, reviews are divided into only positive and negative (based on the scores provided in the reviews themselves), and rottentomatoes.com also uses a binary classification. On the other hand, *metacritic.com* rates everything on a scale from 0-100.

Both of these websites provide annotation guidelines for reviews that don't give pre-assigned numeric ratings, and each of those websites has their editors assign ratings based on their own systems (Metacritic; RottenTomatoes)

Multiple labels - film genres

As your tasks grow in complexity, there are more limiting factors for how to structure your annotations. When facing a task to label movie reviews that only allows one label per document there are a number of ways that the task can be approached, but what happens if it's possible for a document to have more than one label? In Chapter 3 we started discussing a spec for a task involving labeling movie summaries with their associated genres. Let's expand on that example now, in order to see how we can handle more complex annotation tasks.

While it might be tempting to simply say, "Well, we'll only give a single label to each movie", attempting to follow that guideline quickly becomes difficult. Are romantic comedies considered romances, or comedies? You could add "romantic comedy" as a genre label, but will you create a new label for every movie that crosses over a genre line? Such a task quickly becomes ridiculous, simply due to the number of possible combinations. So, define your genres and allow annotators to put as many labels as necessary on each movie (we'll discuss more in (to come) possible approaches to guidelines for such a task).

So how should this information be captured? Of the options listed for the movie review task, some of them can be immediately discarded. Having your annotators change the names of the files to contain the labels is likely to be cumbersome for both the annotators and you: "Casablanca-drama.txt" is easy enough, but "Spaceballs-sciencefiction_comedy_action_parody.txt" would be both annoying for an annotator to create, and equally annoying for you to parse into a more usable form (especially if spelling errors start to sneak in).

Moving files into appropriately labeled folders is also more difficult with this task; a copy of the file would have to be created for each label, and it would be much harder to gather basic information such as how many labels, on average, each movie was given. It would also be much, much harder for annotators to determine if they missed a label.

Earlier in the chapter we represented the positive/negative/neutral information about the film reviews in a table containing a column for the review category with a different movie name in each row. While it would certainly be possible to create a spreadsheet set up the same way to give to your annotators, it's not hard to imagine how error-prone that sort of input would be for a task with even more category options and more potential columns per movie.

So where does that leave us? If none of the more simple ways of labeling data are available, then it's probably time to look at annotation tools and XML representations of annotation data.

In this case, since the information that you want to capture is metadata that's relevant to the entire document, you probably don't need to worry about character offsets, so you can have tags that look like this:

```
<GenreXML>
   <FILM fid = "f1" title = "Cowboys and Aliens" file_name = "film01.txt" />
   <GENRE gid = "g1" filmid = "f01" label = "western" />
   <GENRE gid = "g2" filmid = "f01" label = "sci-fi" />
   <GENRE gid = "g3" filmid = "f01" label = "action" />
</GENREXML>
```

This is a very simple annotation, with an equally simple DTD (If you aren't sure how to read this DTD, go check the DTD sidebar in "What is a DTD?" on page 50):

```
<!ENTITY name "GenreXML">

<!ELEMENT FILM (#PCDATA) >
<!ATTLIST FILM id ID >
<!ATTLIST FILM title CDATA >
<!ATTLIST FILM file_name CDATA >

<!ELEMENT GENRE (#PCDATA) >
<!ATTLIST GENRE id ID >
```

```
<!ATTLIST GENRE filmid CDATA >
<!ATTLIST GENRE label ( action | adventure | classic | … ) >
```

This representation of the genre labeling task is not the only way to approach the problem (in Chapter 3 we showed you a slightly different spec for the same task. Here, we have two elements, film and genre, each with an ID number and relevant attributes. Here, the genre element is linked to the film it represents by the "filmid" attribute.

Don't fall into the trap of thinking there is One True Spec for your task. If you find that it's easier to structure your data in a certain way, or to add or remove elements or attributes, do it! Don't let your spec get in the way of your goal.

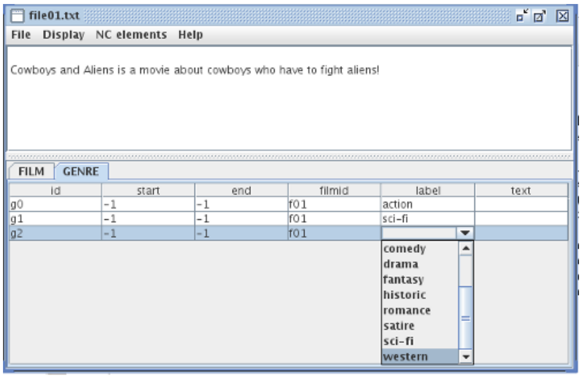
By having the file name stored in the XML for the genre listing, it's possible to keep the annotation completely separate from the text of the file being annotated. We discussed in Chapter ??? the importance of stand-off annotation , and all of our examples will use that paradigm. However, clearly the file_name attribute is not one that is required, and probably not one that you would want an annotator to fill in by hand. But it is useful, and would be easy to generate automatically during pre- or post-processing of the annotation data.

Giving each tag an ID number (rather than only the FILM tags) may not seem very important right now, but it's a good habit to get into because it makes discussing and modifying the data much easier, and can also make it easier to expand your annotation task later if you need to.

At this point you may be wondering how all this extra stuff is going to help with your task. There are a few reasons why you should be willing to take on this extra overhead:

- Having an element that contains the film information allows the annotation to be kept either in the same file as the movie summary, or elsewhere without losing track of the data.
- Keeping data in a structured format allows you to more easily manipulate it later. Having annotation take the form of well-formated XML can make it much easier to analyze later.
- Being able to create a structured representation of your spec helps cement your task, and can show you where problems are in how you are thinking about your goal.
- Representing your spec as a DTD (or other format) means that you can use annotation tools to create your annotations. This can help cut down on spelling and other user-input errors.

Figure Figure 4-2 shows what the film genre annotation task looks like in MAE, an annotation tool that requires only a DTD-like document to set up and get running. As you can see, by having the genre options supplied in the DTD, an annotator has only



Cowboys and Aliens is a movie about cowboys who have to fight aliens!

id	start	end	filmid	label	text
g0	-1	-1	f01	action	
g1	-1	-1	f01	sci-fi	
g2	-1	-1	f01		

Figure 4-2. Genre Annotation in MAE

to create a new instance of the GENRE element and select the attribute they want from the list.

The output from this annotation process would look like this:

```
<FILM id="f0" start="-1" end="-1" text="" title="Cowboys and Aliens" />
<GENRE id="g0" start="-1" end="-1" text="" label="action" />
<GENRE id="g1" start="-1" end="-1" text="" label="sci-fi" />
<GENRE id="g2" start="-1" end="-1" text="" label="western" />
```

There are a few additional elements here than the ones specified in the DTD above—most tools will require certain parameters be met in order to work with a task, but in most cases those changes are superficial. In this case, since MAE is usually used to annotate parts of the text rather than create meta tags, the DTD had to be changed in order to allow MAE to make “GENRE” and “FILM” non-consuming tags. That’s why the “start” and “end” elements are set to -1, to indicate that the scope of the tag isn’t limited to certain characters in the text. You’ll notice that here the filmid attribute in the GENRE tag is not present, and neither is the file_name attribute in the FILM tag. While it wouldn’t be unreasonable to ask your annotators to assign that information themselves, it would be easier to do so with a program—both faster and more accurate.

If you’re planning on keeping the stand-off annotation in the same file as the text that’s being annotated, then you might not need to add the file information to each of the tags. However, annotation data can be a lot easier to analyze/manipulate if it doesn’t have to be extracted from the text it’s referring to, so keeping your tag information in different files that refer back to the originals is generally best practice.

Text Extent Annotation: Named Entities

The review classification and genre identification tasks are examples of annotation labels that refer to the entirety of a document. However, there are many annotation tasks that require a finer-grained approach, where tags are applied to specific areas of the text, rather than all of it at once. We've already discussed many examples of this type of task: part-of-speech tagging, named entity recognition, the time and event identification parts of TimeML, and so on. Basically, any annotation project that requires sections of the text to be given distinct labels falls into this category. We will refer to this as *extent annotation*, because it's annotating a text extent in the data that can be associated with character locations.

In Chapter 3 we discussed the differences between standoff and in-line annotation, and text extents are where the differences become important. The metadata-type tags used for the document classification task could contain start and end indicators or could leave them out; their presence in the annotation software was an artifact of the software itself, rather than a statement of best practice. However, with stand-off annotation it is required that locational indicators are present in each tag. Naturally, there are multiple ways to store this information, such as:

- In-line annotation
- Stand-off annotation by location in a sentence or paragraph
- Stand-off annotation by character location

In the following sections we will discuss the practical applications of each of these methods, using named entity annotation as a case study.

As we discussed previously, named entity annotation aims at marking up what you probably think of as proper nouns— objects in the real world that have specific designators, not just generic labels. So, "The Empire State Building" is a named entity, while "the building over there" is not. For now, we will use the following spec to describe the named entity task:

```
<!ENTITY name "NamedEntityXML">

<!ELEMENT NE (#PCDATA) >
<!ATTLIST NE id ID >
<!ATTLIST NE type ( person | title | country | building | business | …) >
<!ATTLIST NE note CDATA >
```

In-line annotation

While we still strongly recommend not using this form of data storage for your annotation project, the fact remains that it is a common way to store data. The phrase "In-line annotation" refers to the annotation XML tags being present in the text being annotated, and physically surrounding the extent that the tag refers too, like this:

<NE id="i0" type="building">The Massachusetts State House</NE> in <NE id="i1" type="city">Boston, MA</NE> houses the offices of many important state figures, including <NE id="i2" type="title">Governor</NE> <NE id="i3" type="person">Deval Patrick</NE> and those of the <NE id="i4" type="organization">Massachusetts General Court</NE>.

If nothing else, this format for annotation is extremely difficult to read. But more importantly it changes the formatting of the original text. While in this small example there may not be anything special about the text's format, the physical structure of other documents may well be important for later analysis, and in-line annotation makes that difficult to preserve or reconstruct. Additionally, if this annotation were to later be merged with, for example, part-of-speech tagging, the headache of getting the two different tagsets to overlap could be enormous.

Not all forms of in-line annotation are in XML format. There are other ways of marking up data that is inside the text, such as using parenthesis to mark syntactic groups, as was done in Penn TreeBank II (example taken from "The Penn TreeBank: Annotating predicate argument structure" Marcus et al., 1994):

```
(S (NP-SUBJ I
   (VP consider
       (S (NP-SUBJ Kris)
          (NP-PRD a fool))))
```

There are still many programs that provide output in this or a similar format (the Stanford Dependency Parser, for example), and if you want to use tools that do, you may find it necessary to find a way to convert information in this format to stand-off annotation to make it maximally portable to other applications.

Of course, there are some benefits to in-line annotation: it becomes unnecessary to keep special track of the location of the tags or the text that they are surrounding, because those things are inseparable. Still, these benefits are fairly short-sighted, and we strongly recommend not using this paradigm for annotation.

Another kind of in-line annotation is commonly seen in part-of-speech tagging, or other tasks where a label is assigned to only one word (rather than span many words). In fact, you have already seen an example of it in Chapter 1, during the discussion of the Penn TreeBank.

"/" From/IN the/DT beginning/NN ,/, it/PRP took/VBD a/DT man/NN with/IN extraordinary/JJ qualities/NNS to/TO succeed/VB in/IN Mexico/NNP ,/, "/" says/VBZ Kimihide/NNP Takimura/NNP ,/, president/NN of/IN Mitsui/NNS group/NN 's/POS Kensetsu/NNP Engineering/NNP Inc./NNP unit/NN ./.

Here, each part-of-speech tag is appended as a suffix directly to the word it is referring to, without any XML tags separating the extent from its label. Not only does this form of annotation make the data difficult to read, but it also changes the composition of the words themselves. Consider how "group's" becomes "group/NN 's/POS"—the possessive "'s" has been separated from "group", now making it even more difficult to reconstruct the original text. Or, imagine trying to reconcile an annotation like this one with the Named Entity example in the section above! It would not be impossible, but it could certainly cause headaches.

While we don't generally recommend using this format either, many existing part-of-speech taggers and other tools were originally written to provide output in this way, so it is something you should be aware of, as you may need to re-align the original text with the new part-of-speech tags.

We are not, of course, suggesting that you should never use tools that output information in formats other than some variant stand-off annotation. Many of these tools are extremely useful and provide very accurate output. However, you should be aware of problems that might arise from trying to leverage them.

Another problem with this annotation format is that if it is applied to the Named Entity task, there is the immediate problem that the NE task requires that a single tag apply to more than one word *at the same time*. There is an important distinction between applying the same tag more than once in a document (as there is more than one NN tag in the Penn TreeBank example), and applying one tag across a span of words. Grouping a set of words together by using a single tag tells the reader something about that group that having the same tag applied to each word individually does not. Consider these two examples:

<NE id="i0" type="building">The Massachusetts State House</NE> in <NE id="i1" type="city">Boston, MA</NE> ...	The/NE_building Massachusetts/NE_building State/NE_building House/NE_building in Boston/NE_city ,/NE_city MA/NE_city ...

In the example on the left, it is clear that the phrase “The Massachusetts State House” is one unit as far as the annotation is concerned—the NE tag applies to the entire group. On the other hand, in the example on the right the same tag is applied individually to each token, which makes it much harder to determine if each token is a named entity on its own, or if there is a connection between them. In fact, we end up tagging some tokens with the wrong tag! Notice that the state "MA" has to be identified as "/NE_city" in order for the span to be recognized as a city.

Stand-off annotation by tokens

One method that is sometimes used for stand-off annotation is done by *tokenizing* (i.e., separating) the text input and giving each token a number. The tokenization process is usually based on whitespace and punctuation, though the specific process can vary by program (e.g., some programs will split “'s” or “n’t” from “Meg’s” and “don’t”, and others will not). The text in the appended annotation example has been tokenized—each word and punctuation mark have been pulled apart.

Taking the above text as an example, there are a few different ways to identify the text by assigning numbers to the tokens. One way is to simply number every token in order, starting at 1 (or 0, if you prefer) and going until there are no more tokens left:

Table 4-1. Word labeling by token

TOKEN	TOKEN_ID

"	1
From	2
the	3
beginning	4
,	5
…	…
unit	31
.	32

This data could be stored in a tab-separated file or in a spreadsheet, as it's necessary to keep the IDs associated with each token. Another way is to assign numbers to each sentence, and identify each token by sentence number and its place in that sentence:

Table 4-2. Word labeling by sentence and token

TOKEN	SENT_ID	TOKEN_ID
"	1	1
From	1	2
the	1	3
beginning	1	4
,	1	5
…		
unit	1	31
.	1	32
Then	2	1
…		

Naturally, more identifying features could be added, such as paragraph number, document number, and so on. The advantage of having additional information (such as sentence number) used to identify tokens is that this information can be used later to help define features for the machine learning algorithms (while sentence number could be inferred again later, if it's known to be important then it's easier to have that information up front.

Annotation data using this format could look something like this:

Table 4-3. Part of speech annotation in tokenized text

POS_TAG	SENT_ID	TOKEN_ID
"	1	1
IN	1	2

DT	1	3
NN	1	4
…		

There are some advantages to using this format: because the annotation is removed from the text, it's unnecessary to worry about overlapping tags when trying to merge annotations done on the same data. Also, this form of annotation would be relatively easy to set up with a tokenizer program and any text that you want to give it.

However, there are some problems with this form of annotation as well. As you can see, because the text is split on whitespace and punctuation the original format of the data cannot be recovered, so the maxim of "do no harm" to the data has been violated. If the structure of the document that this text appeared in later became important when creating features for a classifier, it could be difficult to merge this annotation with the original text format.

It is possible to use token-based annotation without damaging the data, though it would require running the tokenizer each time the annotation needed to be paired with the text, and the same tokenizer would always have to be used. This is the suggested way for dealing with token-based standoff annotation.

Additionally, this format has a similar problem to the appended annotation, in that it appears to assume that each tag applies to only one token. While it's not impossible to apply a tag to a set of tokens, the overhead does become greater. Consider again our Named Entity example, this time tokenized:

TOKEN	**SENT_ID**	**TOKEN_ID**
The	1	1
Massachusetts	1	2
State	1	3
House	1	4
in	1	5
Boston	1	6
,	1	7
MA	1	8
houses	1	9
…		

And here is how we would apply a tag spanning multiple tokens:

TAG	START_SENT_ID	START_TOKEN_ID	END_SENT_ID	END_TOKEN_ID
NE_building	1	1	1	4
NE_city	1	6	1	8

The other flaw in this method is that it doesn't easily allow for annotating parts of a word. Annotation projects focusing on morphemes or verb roots would require annotating partial tokens, which would be difficult with this method. It isn't impossible to do—another set of attributes for each token could be used to indicate which characters of the token are being labeled. However, at that point one might as well move to character-based stand-off annotation, which we will discuss in the next section.

Stand-off annotation by character location

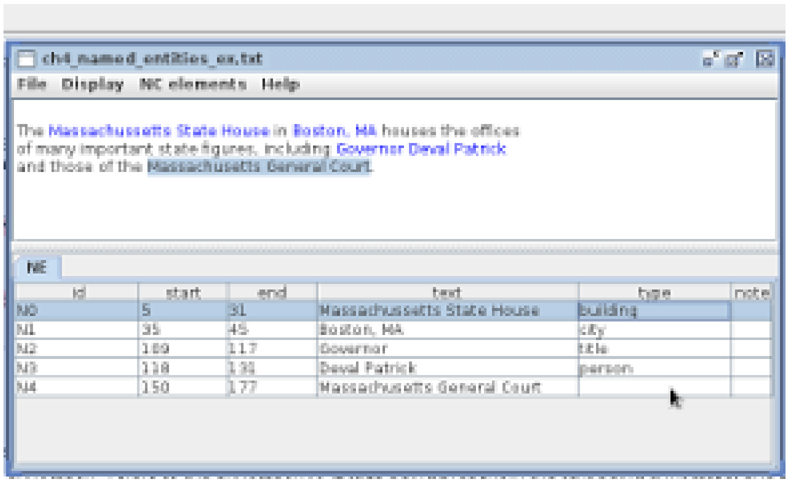
Using character locations to define what part of the document a tag applies to is a reliable way to generate a standoff annotation that can be used across different systems. Character-based annotations use the character offset information to place tags in a document, like this:

```
The Massachussetts State House in Boston, MA houses the offices
of many important state figures, including Governor Deval Patrick
and those of the Massachusetts General Court.

<NE id="N0" start="5" end="31" text="Massachussetts State House" type="building" />
<NE id="N1" start="35" end="45" text="Boston, MA" type="city" />
<NE id="N2" start="109" end="117" text="Governor" type="title" />
<NE id="N3" start="118" end="131" text="Deval Patrick" type="person" />
<NE id="N4" start="150" end="177" text="Massachusetts General Court" type="organization" />
```

At fist glance it is difficult to see the benefits to this format for annotation—the start and end numbers don't mean much to someone just looking at the tags, and the tags are so far from the text as to be almost unrelated. However, this distance is precisely why stand-off annotation is important, even necessary. By separating the tags from the text, it becomes possible to have many different annotations pointing to the same document without interfering with each other, and more importantly, without changing the original text of the document.

As for the start and end numbers, while they are difficult for a human to determine what they are referring to, it's very easy for computers to count the offsets to find where each tag is referring to. And the easier it is for the computer to find the important parts of text, the easier it is to use that text and annotation for machine learning later.



The Massachussetts State House in Boston, MA houses the offices
of many important state figures, including Governor Deval Patrick
and those of the Massachusetts General Court

id	start	end	text	type	note
N0	5	31	Massachussetts State House	building	
N1	35	45	Boston, MA	city	
N2	109	117	Governor	title	
N3	118	131	Deval Patrick	person	
N4	150	177	Massachusetts General Court		

Figure 4-3. Named entity annotation

Using "start" and "end" as attribute names to indicate where the tags should be placed in the text is a convention that we use here, but is not one that is a standard in annotation—different annotation tools and systems will use different terms for this information. Similarly, the "text" attribute does not have any special meaning either. What the attributes are called is not important; what's important is the information that they hold.

Technically, all that's needed for these tags to work are the start and end offset locations and the tag attributes—here, the tags also contain the text that the tag applies to because it make the annotation easier to evaluate. Even if that information was not there, the tag would still be functional. Here's what it might look like to create this annotation in an annotation tool:

Naturally, some preparation does have to be done in order to make stand-off annotation work well. For starters, it's important to early on decide on what character encoding you will use for your corpus, and to stick with that throughout the annotation process. The character encoding that you choose will determine how different computers and programs count where the characters are in your data, and changing encodings partway through can cause a lot of work to be lost. We recommend using UTF-8 encoding for your data.

Encoding problems can cause a lot of headaches, especially if your data will be transferred between computers using different operating systems. Using Windows can make this particularly difficult, as it seems that Windows does not default to using UTF-8 encoding, while most other operating systems (Mac and most flavors of Unix/Linux, that we're aware of) do. It's not impossible to use UTF-8 on Windows, but it does require a little extra effort.

Linked Extent Annotation: Semantic Roles

Sometimes in annotation tasks it is necessary to represent the connection between two different tagged extents. For example, in temporal annotation it is not enough to annotate "Monday" and "ran" in the sentence "John ran on Monday"; in order to fully represent the information presented in the sentence we must also indicate that there is a connection between the day and the event. This is done by using relationship tags, also called link tags.

Let's look again at our example sentence about Boston. If we were to want to add locational information to this annotation, we would want a way to indicate that there is a relationship between places. We could do that by adding a tag to our DTD that would look something like this:

```
<!ELEMENT L_LINK EMPTY >
<!ATTLIST L-LINK fromID IDREF >
<!ATTLIST L-LINK toID IDREF >
<!ATTLIST L-LINK relationship ( inside | outside | same | other )
```

Obviously, this is a very limited set of location relationships, but it will work for now. How would this be applied to the annotation that we already have?

This is where the tag IDs that we mentioned in Section "Multiple labels - film genres" on page 70 become very important. Because link tags do not refer directly to extents in the text, they need to represent the connection between two annotated objects. The most common way to represent this information is to use the ID numbers from the extent tags to anchor the links. This new information will look like this:

```
he Massachussetts State House in Boston, MA houses the offices
of many important state figures, including Governor Deval Patrick
and those of the Massachusetts General Court.

<NE id="N0" start="5" end="31" text="Massachussetts State House" type="building" />
<NE id="N1" start="35" end="45" text="Boston, MA" type="city" />
<NE id="N2" start="109" end="117" text="Governor" type="title" />
<NE id="N3" start="118" end="131" text="Deval Patrick" type="person" />
<NE id="N4" start="150" end="177" text="Massachusetts General Court" type="organization" />

<L-LINK id="L0" fromID="N0" toID="N1" relationship="inside" />
<L-LINK id="L0" fromID="N4" toID="N0" relationship="inside" />
```

By referring to the IDs of the Named Entity tags, we can easily encode information about the relationships between them. And because the L-LINK tags also have ID numbers, it is possible to create connections between them as well—perhaps a higher level of annotation could indicate that two L-LINKS represent the same location information, which could be useful for a different project.

Once again, the names of the attributes here are not of particular importance. We use "fromID" and "toID" as names for the link anchors because that is what the annotation tool MAE does, but other software uses different conventions. The intent, however, is the same.

ISO Standards and you

In "ISO Standards" on page 60 we discussed the LAF (Linguistic Annotation framework) standard for representing annotated data. It might have sounded pretty formal, but don't worry! If you're following along with our recommendations in this book and using XML-based standoff annotation, chances are that your annotation structure is already LAF-compliant, and that you would just need to convert it to the LAF dump format. Also keep in mind: LAF is a great foundation for linguistic researchers who want to share their data, but if your annotation is only meant for you, or is proprietary to a company, this might not be something that you will need to worry about at all.

Summary

- There are many different ways that annotations can be stored, but it's important to choose a format that will be flexible and easy to change later if you need to. We recommend stand-off, XML-based formats.
- In some cases, such as single-label document classification tasks, there are many ways to store annotation data, but these techniques are essentially isomorphic. In such cases, choose which method to use by considering how you are planning to use the data, and what methods work best for your annotators.
- For most annotation tasks, such as those requiring multiple labels on a document, and especially those require extent annotation and linking, it will be useful to have annotation software for your annotators to use. See Appendix ??? for a list of available software.
- Extent annotation can take many forms, but character-based stand-off annotation is the format that will make it easier to make any necessary changes to the annotations later, and also make it easier to merge with other annotations.
- If you do choose to use a character-based stand-off annotation, be careful about what encodings you use for your data, especially when you create the corpus in the first place. Different programming languages and operating systems have different default settings for character encoding, and it's vital that you use a format that will work for all your annotators (or at least be willing to dictate what resources your annotators use).

- Using industry standards like XML, and annotation standards like LAF for your annotations will make it much easier for you to interact with other annotated corpora, and make it easier for you to share your own work.

APPENDIX

Bibliography

James Allen. *Towards a general theory of action and time*. Artificial Intelligence 23, pp. 123-154. 1984.

Sue Atkins and N. Ostler. *Predictable meaning shift: some linguistic properties of lexical implication rules* in Lexical Semantics and Commonsense Reasoning. J. Pustejovsky & S. Bergler, Eds. Springer-Verlag, NY. pp. 87-98. 1992.

Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2010.

Noam Chomsky. *Syntactic Structures*. Mouton, 1957.

W.N. Francis and H. Kucera. *Brown Corpus Manual*. Available online: http://khnt.aksis.uib.no/icame/manuals/brown/ accessed Aug. 2011. 1964, revised 1971 and 1979.

Patrick Hanks and James Pustejovsky. *A Pattern Dictionary for Natural Language Processing*. Revue Française de Linguistique Appliquee. 2005.

Livnat Herzig, Alex Nunes and Batia Snir. *An Annotation Scheme for Automated Bias Detection in Wikipedia*. In the Proceedings of the 5th Linguistic Annotation Workshop. 2011.

Nancy Ide, Laurent Romary. *International standard for a linguistic annotation framework*. Journal of Natural Language Engineering, 10:3-4, 211-225. 2004.

Nancy Ide, Laurent Romary. *Representing Linguistic Corpora and Their Annotations*. Proceedings of the Fifth Language Resources and Evaluation Conference (LREC), Genoa, Italy. 2006.

Thomas Kuhn. *The Function of Measurement in Modern Physical Science*. The University of Chicago Press, 1961.

Geoffrey Leech. *The state of the art in corpus linguistics*, in English Corpus Linguistics: Linguistic Studies in Honour of Jan Svartvik, London: Longman, pp.8-29. 1991.

Mitchell P. Marcus , Beatrice Santorini , Mary Ann Marcinkiewicz. *Building a Large Annotated Corpus of English: The Penn Treebank*. Computational Linguistics, 19:2; pgs 313-330. 1993

James Pustejovksy. "*Unifying Linguistic Annotations: A TimeML Case Study*." In *Proceedings of Text, Speech, and Dialogue Conference* 2006.

James Pustejovsky, José Castaño, Robert Ingria, Roser Saurí, Robert Gaizauskas, Andrea Setzer and Graham Katz. 2003. *TimeML: Robust Specification of Event and Temporal Expressions in Text*. IWCS-5, Fifth International Workshop on Computational Semantics.

James Pustejovsky, Patrick Hanks, Anna Rumshisky. *Automated Induction of Sense in Context*. ACL-COLING, Geneva, Switzerland. 2004.

David A. Randell, Zhan Cui, & Anthony G. Cohn. *A spatial logic based on regions and Q17 connections*. In M. Kaufmann (Ed.), Proceedings of the 3rd international 905 conference on knowledge representation and reasoning. San Mateo, CA. (pp. 165–176). 1992.

Randolph Quirk, Sidney Greenbaum, Geoffrey Leech, Jan Svartik. *A Comprehensive Grammar of the English Language*. Longman, 1985