The Language of Natural Language Processing (NLP)

The Basic Language of NLP

Computational linguistics relies on a standard terminology you should become familiar with, in particular, lemmatization and lemmas, tokenization and sentence splitting, part-of-speech tagging (POS), named-entity recognition (NER), topic modeling (or document classification), but also distributional semantics and word sense disambiguation.

Sentence Splitting & sentences

Sentence splitting refers to the operation of marking the start and end of a sentence. If word tokenization splits a text into words and punctuation marks, sentence splitting assembles the tokenized text into sentences.

Recognizing the end of a sentence is not an easy task for a computer. In English, punctuation marks that usually appear at the end of a sentence may not indicate the end of a sentence. The period is the worst offender. A period can end a sentence but it can also be part of an abbreviation or acronym, an ellipsis, a decimal number, or part of a bracket of periods surrounding a Roman numeral. A period can even act both as the end of an abbreviation and the end of a sentence at the same time. And some poems may not contain any sentence punctuation at all.

Like tokenization, sentence splitting is unlikely to be of direct interest to human or social scientists. But like tokenization, sentence splitting is a prerequisite of all other NLP operations.

Tokenization & words (Word Boundaries)

Tokenization tools split a sentence into its constitutive single words. It is one of the most basic NLP tools, a prerequisite of all other NLP operations.

Lemmatization & lemmas

Lemmatization serves the purpose of reducing the range of words in the corpus analyzed in view of their automatic analysis. Word frequency distributions and any graphical display of information should ideally be based on lemmatized words.

A lemmatizer turns the different *inflection* of a verb (killed, kills, kill) to its infinitive form (i.e., kill) and nouns (e.g., workers, worker) to its singular form (i.e., worker) (male for many languages).

Often, one can also talk about **stemming**, instead of lemmatizing. While lemmatizing keeps the standard dictionary base form, stemming will reduce the different inflections to their common root (e.g., killed, kills, kill, killer, killers, killing to kill); the stem may not necessarily be a dictionary term in the language.

Good freeware lemmatizers for the English language (but some can process other languages as well) are the Stanford lemmatizer, TreeTagger, AntCon, LemmaGen.

Lemmatization programs require software training but in most programs this has been done by the software developers for different languages. Of course, the training corpus is typically current newswire articles. Depending upon the domain you need to analyze (e.g., 19th century documents) this may not provide the best results for your corpus.

Stopwords

Stopwords are any word in a list of words which are filtered out (i.e., stopped) before or after processing of natural language data (text). There is no single universal list of stop words used by all NLP tools, nor any agreed upon rules for identifying stop words. Typically, pronouns, determiners, auxiliaries, adverbs are included in the list. But, any group of words can be chosen as the stop words for a given purpose. As Manning et al. (2008: 27) write: “[The] general trend in [information retrieval] systems over time has been from standard use of quite large stop lists (200–300 terms) to very small stop lists (7–12 terms) to no stop list whatsoever.”

Sometimes, stopwords are also called **junk words** or **function words**, as opposed to **content words** that provide the bulk of meaning in a text.

Caveat. Pennebaker’s work on function words show the importance these words have in detecting people’s writing style and psychological state (Pennebaker and King 1999; Pennebaker et al. 2003; Chung and Pennebaker 2007).

For more information, history, and scholarly references, see the good **Wikipedia** page https://en.wikipedia.org/wiki/Stop_word#cite_note-2.

You will find a couple of lists of common stopwords under the lib\wordLists subdirectories: stopwords.txt and sw.txt.

Part of Speech Tagging (POS)

POS annotates automatically every single word in a corpus by syntactical category, i.e., classifying each word as a noun, adjective, verb, adverb... Thus, the sentence, “they have eaten”, would be tagged as: “they” pronoun, “have” auxiliary, and “eaten” verb.
It is likely that POS tools are of less interest to social scientists, except that POS is used in many other NLP components (e.g., parsing, and even lemmatizing). However, **you may still be interested in extracting from your corpus just verbs or nouns and then get a frequency distribution of just those parts of speech.** A simple frequency distribution of words would give you that same information, of course, but not by part of speech (the word “assault” could be a verb or a noun).

Dependency Relations Tagging (DEPREL)

The Stanford CoreNLP parser provides typed dependencies representations (i.e., a simple description of the grammatical relationships), dependency relations (**DEPREL**) between words in a sentence, e.g., whether a noun is subject or object, an active or passive subject.

Named-Entity Recognition (NER)

Named-entity recognition (NER) (also known as **entity identification**, **entity chunking** and **entity extraction**) seeks to locate and classify elements in a text into pre-defined categories (known as the NER model). NER models can be based on **three categories** (location, person, organization), **four categories** (location, person, organization, other), and **seven categories** (location, date, time, person, organization, money, percent).

NER would help you with automatic annotation (i.e., coding) of your documents.

There are **two phases** to the process:

1. Training
2. Automatic text processing (lemmatizing, recognizing, …)

Automatic NER tools require some human training. Once trained, they will do the work for you. And, needless to say, the better you train them, the better work they will later produce for you!

1. *Training an NER program.* Using a subset of all the documents in your corpus, you need to annotate the documents in the subset for the categories in the NER model you are using. This annotated file will then serve as input to an NER program for training purposes, i.e., the NER program will “look” at your annotations and learn how to annotate independently text not previously annotated. The NER program will compute an NER model (parameter file) containing generalized information about all the entities it has extracted for the 3, 4, or 7 selected categories; the model can be exported as a file.

2. *Automatic NER*. The second step in NER is to apply the model to process automatically all the non-coded documents, i.e., the raw data of your corpus.

Basically, the two steps of *training* and *processing* have the following sequence of input and output:

Training	ANNOTATED DATA → NER → MODEL
Processing	MODEL + RAW DATA → NER → AUTOMATIC ANNOTATED DATA

Annotated NER files (both input and output) have the same general standard format. Thus, the following sentence:

> Wolff, currently a journalist in Argentina, played with Del Bosque in the final years of the seventies in Real Madrid.

would have the following annotation:

[PER Wolff], currently a journalist in [LOC Argentina], played with [PER Del Bosque] in the final years of the seventies in [ORG Real Madrid].

In this annotation, PER, LOC, ORG stand for the model categories PERson, LOCation, and ORGanization.

The input and output file to/from an NER program would also have a standard structure, with one line per word, each line containing the word itself followed by the alphabetic character B or O and the entity category. Each word in a text either belongs to one of the model categories (then it is marked with B) or does not, in which case it is marked as O, outside the model. When multiple words belonging to a model category are used, but they have the same meaning of a single word (e.g., "commit an assault" which is the exact equivalent of "assault"), each word is then also tagged separately as B (beginning) and I (for inside); thus, "commit" is tagged as B-ACTion, "assault" as I-ACTion. 3rd, 4th, 5th… positioned words are all tagged as I-category.

```
  Wolff B-PER
     , O
 currently O
     a O
journalist O
    in O
 Argentina B-LOC
     , O
   played O
    with O
    Del B-PER
  Bosque I-PER
    in O
```

```
the O
final O
years O
of O
the O
seventies O
in O
Real B-ORG
Madrid I-ORG
. O
```

The output of the second NER step (i.e., the AUTOMATIC ANNOTATED DATA) has the same file format of the ANNOTATED DATA file input to NER in the training phase.

Typically, you divide up the annotated documents (prepared for input for step 1) into two sets: one set (approximately 90%) for *training* and another set (the remaining 10%) for *testing*. In testing, you would use the same 10% of documents *with no annotation.* For this 10% of documents you can compare the results of manual and automatic annotation, hoping for at least an 80% accuracy.

Needless to say, the larger the body of work manually annotated and fed to the system as input in the training phase, and the more accurate the automatic annotation.

A typical training system will be based on 5-to-10 thousand sentences, the more the better.
You can also use an NER program with a pre-defined model derived from a standard set of NER categories from a very large corpus.

Once you have coded some documents in PC-ACE, PC-ACE can create automatically the NER annotated data file and export it for use in an NER program for NER training. PC-ACE can then import back the NER model file created by the NER program. PC-ACE will import this model file as a two-field dictionary usertable containing a record for each dictionary value (i.e., a word, “assault”, or set of single-meaning words, “commit an assault”) and the name of its model category (or simplex in the PC-ACE grammar, for example “action), doing all the necessary conversions. PC-ACE will then use this dictionary usertable for assisted coding of any new document.

Good freeware software options for NER are the Stanford NER, Open NLP Apache, and NLTK.

Parsing/Parsers

Parsing in NLP refers to the process of determining the syntactic structure of a text by analyzing its constituent words based on an underlying grammar (of the language). From the Stanford CoreNLP parser (https://nlp.stanford.edu/software/lex-parser.html) we read: “A natural language parser is a program that works out the grammatical structure of sentences, for instance, which groups of words go together (as "phrases") and which words are the subject or object of a verb. Probabilistic parsers use knowledge of language gained from hand-parsed sentences to try to produce the most likely analysis of new sentences. These statistical parsers still make some mistakes, but commonly work rather well. Their development was one of the biggest breakthroughs in natural language processing in the 1990s."

Stanford CoreNLP

Several different (mostly freeware) tools are available in computational linguistics. But, certainly, the freeware, open source, Stanford CoreNLP, provides the gold standard in NLP. Stanford CoreNLP parser uses a state-of-the-art Probabilistic Context free Grammar (PCFG) (Klein and Manning 2003).

Klein, Dan and Christopher D. Manning. 2003. "Accurate Unlexicalized Parsing." In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pp. 423–430. Association for Computational Linguistics.

For more information, see the TIPS file on NLP software options.

We read on the Stanford CoreNLP website (http://nlp.stanford.edu/software/corenlp.shtml):

> Stanford CoreNLP provides a set of natural language analysis tools which can take raw text input and give the base forms of words, their parts of speech, whether they are names of companies, people, etc., normalize dates, times, and numeric quantities, and mark up the structure of sentences in terms of phrases and word dependencies, indicate which noun phrases refer to the same entities, indicate sentiment, etc. Stanford CoreNLP is an integrated framework. Its goal is to make it very easy to apply a bunch of linguistic analysis tools to a piece of text. Starting from plain text, you can run all the tools on it with just two lines of code. It is designed to be highly flexible and extensible. With a single option you can change which tools should be enabled and which should be disabled. Its analyses provide the foundational building blocks for higher-level and domain-specific text understanding applications.
>
> Stanford CoreNLP integrates many of our NLP tools, including the part-of-speech (POS) tagger, the named entity recognizer (NER), the parser, the coreference resolution system, the sentiment analysis, and the bootstrapped pattern learning tools. The basic distribution provides model files for the analysis of English, but the engine is compatible with models for other languages. Below you can find packaged models for Chinese and Spanish, and Stanford NLP models for German and Arabic are usable inside CoreNLP.
>
> Stanford CoreNLP is written in Java and licensed under the GNU General Public License (v3 or later; in general Stanford NLP code is GPL v2+, but CoreNLP uses several Apache-licensed libraries, and so the composite is v3+). Source is included. Note that this is the full GPL, which allows many free uses, but not its use in proprietary software which is distributed to others. The download is 260 MB and requires Java 1.8+.

In **input**, CoreNLP takes a single, txt-formatted file containing the linguistic corpus you wish to analyze. In **output**, it produces the **CoNLL table** in csv or html format. The CoNLL table is the basis of all computational linguistics analyses (e.g., frequency distribution of words). It is produced as the output (in csv format) of the Stanford CoreNLP, the basic tool of NLP.

For more information, see the TIPS file on Stanford CoreNLP download install run.

The CoNLL table

Parsers produce in output the CoNLL table, the basis of all computational linguistics analyses (e.g., frequency distribution of words).

The CoNLL table contains various information on each word contained in the document, its position in the document, its lemmatized form, syntactical category of the word in the sentence (e.g., noun, verb), Named-Entity Recognition (NER) and more. This information is used to compute statistical analyses on the text, namely frequency distribution of words (particularly lemmatized words but also by different types of syntactical categories, e.g., verbs or nouns), average sentence length, n-grams, co-occurrences).

For more information, see the TIPS file on the CoNLL table.

Topic Modeling (Document Classification)

Topic modeling allows you to get an idea of what the documents talk about and what the differences among documents might be.

A topic model is a type of statistical model for discovering the abstract “topics” that occur in a collection of documents (typically referred to as a “bag of words”). Intuitively, given that a document is about a particular topic, one would expect particular words to appear in the document more or less frequently: “negro” and “white woman” will appear more often in documents about lynching; “strike”, “trade unions”, and “workers” will appear in documents about labor conflict; “the”, “is”, “and” will appear equally in both.

A document typically deals with multiple topics in different proportions; thus, in a document about lynching you may find words and sentences about politics, to the extent that politicians and official state organizations react to a lynching event. A topic model captures these differences mathematically (simple frequency distributions but also more complex mathematical tools such as LDA, Latent Dirichlet Allocation), which allows examining a set of documents and discovering, based on the statistics of the words in each, what the topics might be and what each document’s balance of topics is.

Good **freeware programs** for topic modeling for the English language are the Stanford Topic Modeling Toolbox, Mallet (some can process other languages as well).

Topic modeling programs require software training. But already trained models may also be available.

For more information, see the TIPS file on Mallet Topic Modeling.

Machine learning and Neural network approaches

The tools described above work very well with small text corpora. The world of big data, of millions of words and documents, has brought out a series of new tools based on machine learning (i.e., tools that require training), such as text classifiers as Support Vector Machines (SVM) and Word2Vec, and the new cutting-edge neural network approaches.

References

Chung, Cindy and James Pennebaker. 2007. “The Psychological Functions of Function Words.” In: pp. 343-359, Klaus Fiedler (Ed.), Social Communication, New York: Psychology Press.

Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge: Cambridge University Press.

Pennebaker, James W., Matthias R. Mehl, and Kate G. Niederhoffer. 2003. “Psychological Aspects of Natural Language Use: Our Words, Our Selves.” Annual Review of Psychology, Vol. 54, pp. 547–77.

Pennebaker, James W. and Laura A. King. 1999. “Linguistic Styles Language Use as an Individual Difference,” *Journal of Personality and Social Psychology*, Vol. 77, No.6, pp. 1296-1312.