# So, you need to understand language data? Open-source NLP software can help!

Understanding language is not easy, even for us humans, but computers are slowly getting better at it. 50 years ago, the psychiatrist chat bot Elyza could successfully initiate a therapy session but very soon you understood that she was responding using simple pattern analysis. Now, the IBM's supercomputer Watson defeats human champions in a quiz show live on TV. The software pieces required to understand language, like the ones used by Watson, are complex. But **believe it or not, many of these pieces are actually available for free as open-source**. This post summarizes how open-source software can help you analyze language data using this flow chart as a guideline.

    ![Slide 1]({ base.url [1]}/assets/white-paper/slide1.png)

If your language data is already available as **text**, it is most likely to be stored in files. Apache libraries like POI and PDFBox extract text from the most common formats. Apache Tika is a toolkit that uses such libraries to extract text and other types of metadata from most types of documents. It has the additional feature of identifying the document language if needed. A great way of organizing and accessing text data is by using a search engine like Apache Solr, which also comes as open-source.

If your language data is on the web, most likely it is part of some other larger website. BoilerPipe helps discard irrelevant text like navigational menu or ads.

If your language data is stored in **images** (note that often PDFs are also simply images of text), you need to use OCR to extract the text. Tesseract can work well out of the box, or can be tuned to get commercial grade quality.

Finally, if your language data is stored as **audio**, you should try CMU Sphinx for converting speech to text. Depending on how diverse the speakers are and what they say, the results may be quite usable. I must add that the support CMU provides via email, forums and IRC is actually better than the support most commercial software vendors offer.

Independently on where the language data comes from, it most likely is what we NLP people call "dirty" and requires cleaning. It may have some idiosyncrasies that will prevent libraries trained on clean text to work well. Cleaning isn't complex work, but it can be time consuming. It's best to automate it with custom-written scripts. It's a bit like with painting a room: the more time you spend preparing, the better the end result.

Now, it all depends on what exactly you need to find out about your text, and what kind of data you already have. For example, methods that fall under

**text classification** (also: text categorization) can be quite versatile and powerful. They can be used for detecting spam, guessing genre, estimating overall attitude or sentiment, and can even predict characteristics of the text author, like their age and gender. But these methods also rely on two assumptions: a) the classes are known in advance and b) training data available for each of these classes. The quality of a text classification approach always depends on the number of examples you are training on and how distinct are the classes. Toolkits like NLTK and LingPipe offer text classification as one of their features. But in fact, most machine learning libraries have features that makes them easily applicable to text, e.g. Weka. There is also LibShortText, useful when you analyze short text like tweets.

If you have a lot of data, although none is labeled with classes and in fact you have no idea what classes the data contains, try **topic modeling**, which clusters documents, while providing a series of possible labels (topics) for each cluster. Mallet offers both text classification and topic modelling. Gensim is a scalable topic modeling library in Python.

One step deeper in terms of what we can derive from text, is **keyword extraction** (also: tagging, keyphrase or subject indexing) Compared to text classification and topic modeling, keyword extraction assumes that the number of possible topics (or categories, or tags) can be large (thousands or more), that each document may have a different number of matching topics and that these topics are well-formed phrases. Keywords can be chosen from document text or from a predefined vocabulary, which ensures their consistency. Kea and Maui both extract keywords, but use somewhat different techniques.

Whereas text classification, topic modeling and keyword extraction all summarize the content of the document categorically, it is also possible to

summarize by extracting the most relevant complete sentences from text. A **text summarization** platform MEAD allows one to try out different summarization algorithms and evaluate their performance using standard metrics.

Going beyond key categories and phrases, one could extract all those concepts and entities mentioned in text and identify their meaning. **Entity linking** performs this by disambiguating entities to their unique ids in a knowledge base. For example, a technique called wikification links phrases appearing in text to articles in Wikipedia. Wikipedia Miner and Illinois Wikifier both specialize in this task.

If you are interested specifically in names of people, organizations and locations, **entity recognition** (also: named entity tagging, NER) is the technique to use. Illinois Named Entity Tagger, OpenNLP and Stanford NLP all perform this task. If you are not working with documents similar to news articles, make sure to annotate some data by hand and train these tools first.

Going deeper we can determine what the text actually says, i.e. what are the facts expressed in its sentences. When people speak or write, they use pronouns and other ways to refer to the same entities in order to avoid repetition and sound nice. The computer first needs to understand which pronoun and expression refers to which entity. The technique to do this is called **coreference resolution** (also: anaphor resolution). The same three entity recognition libraries offer these capabilities.

The next step is to determine how the entities are connected semantically and syntactically. This is called **parsing**. Stanford NLP is known for its statistical parser that works in many languages, but other libraries, like MaltParser and OpenNLP also offer their versions.

The majority of NLP software is available as a set of Java or Python toolkits. If neither of these langauges is your thing, look out for ports to other languages, such as the Stanford.NLP.Net in F#.

One advantage of using toolkits is that they make it easy to pass the output from one NLP component to another. However, sometimes, you need to combine components from different libraries. UIMA and GATE both mitigate this problem by offering frameworks, which can combine components from different authors, some of which can be open-source and others commercial, into a single systems. Another way to do this, is to use the NLP Interchange Format (NIF) which connects NLP components using an RDF/OWL format for encoding their input/output.

**So, why is it possible for such complex software to be free?** Researchers at universities are spending decades improving performance of many individual components and publish their results in conference papers. Releasing the software as open-source allows them to improve the state-of-the-art by collaborating on the same task over time and across universities, benchmark solutions against each other and using smaller components to build more complex NLP systems. It also gets researchers more citations. Everyone benefits.