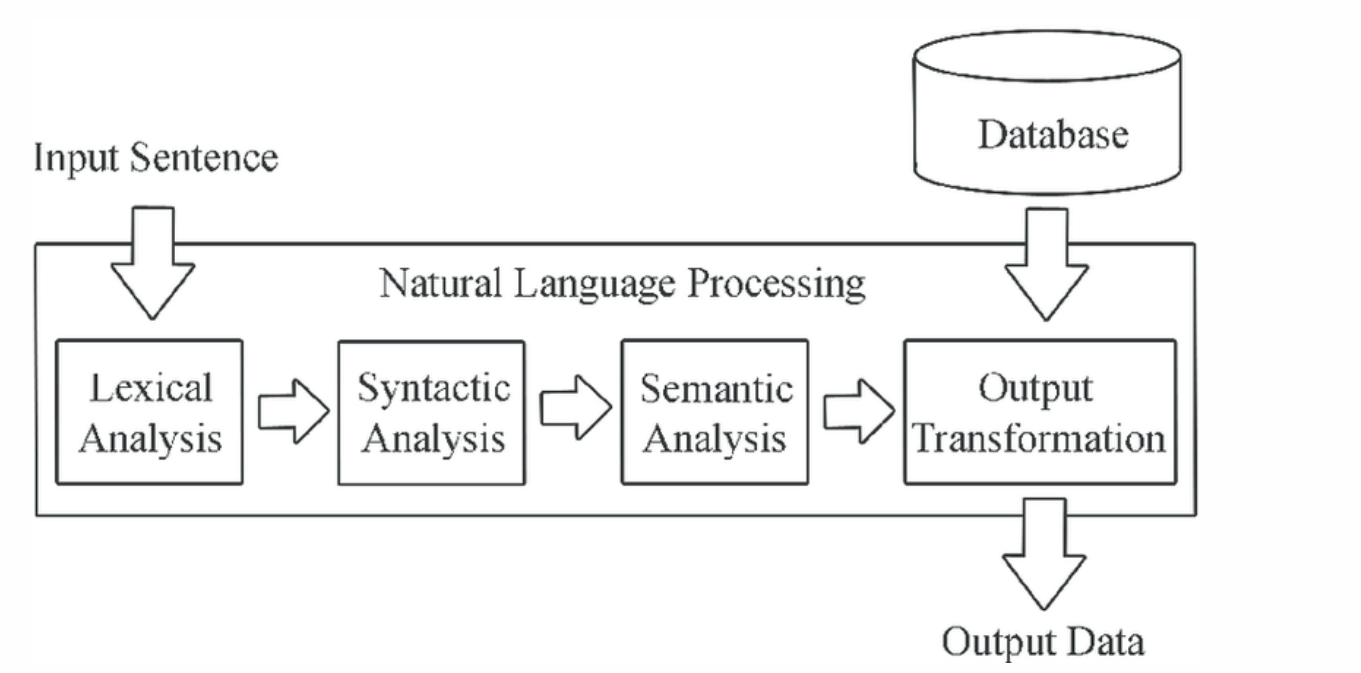
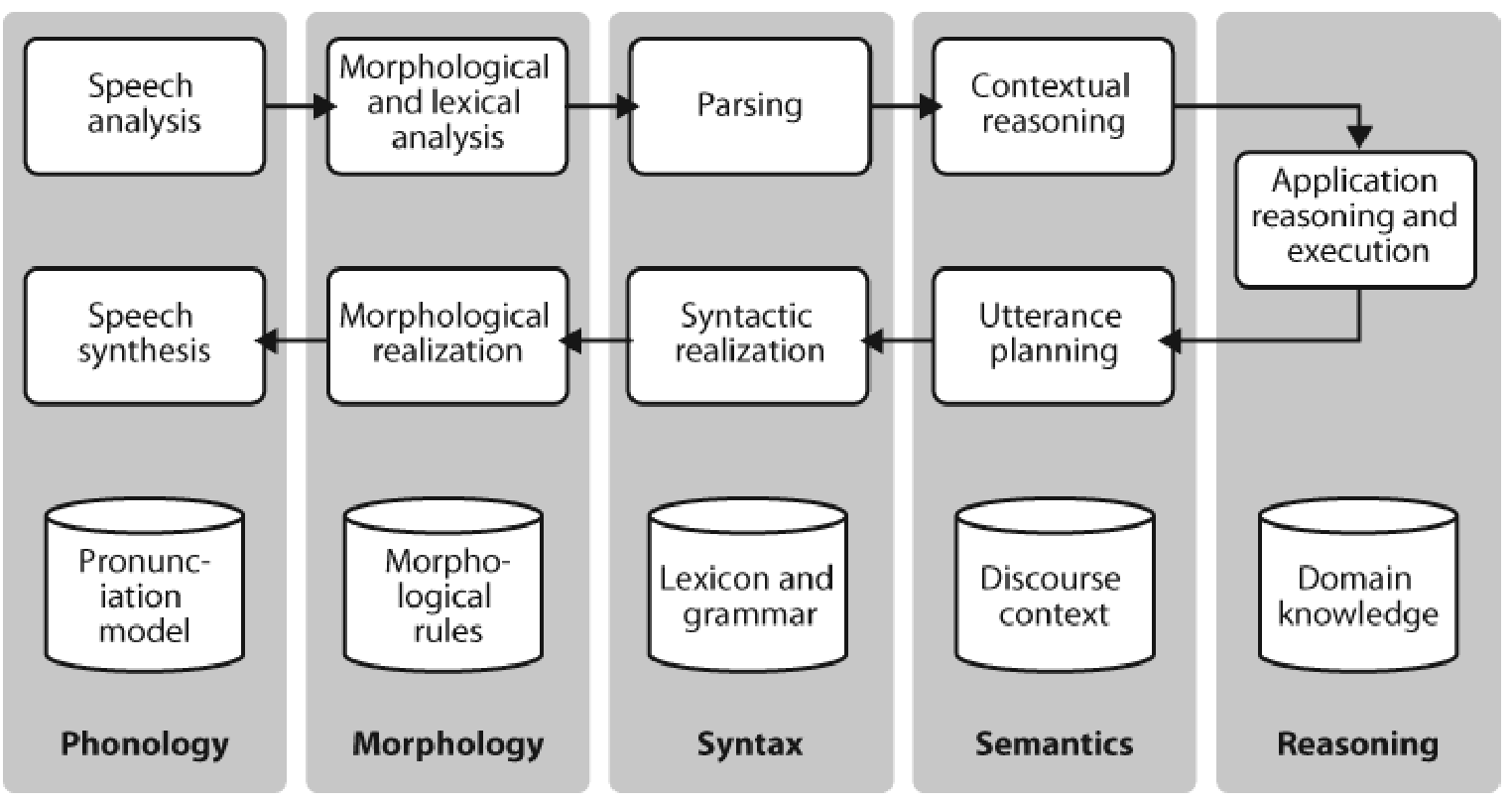


****

****

**lexicon, or lexical resource**

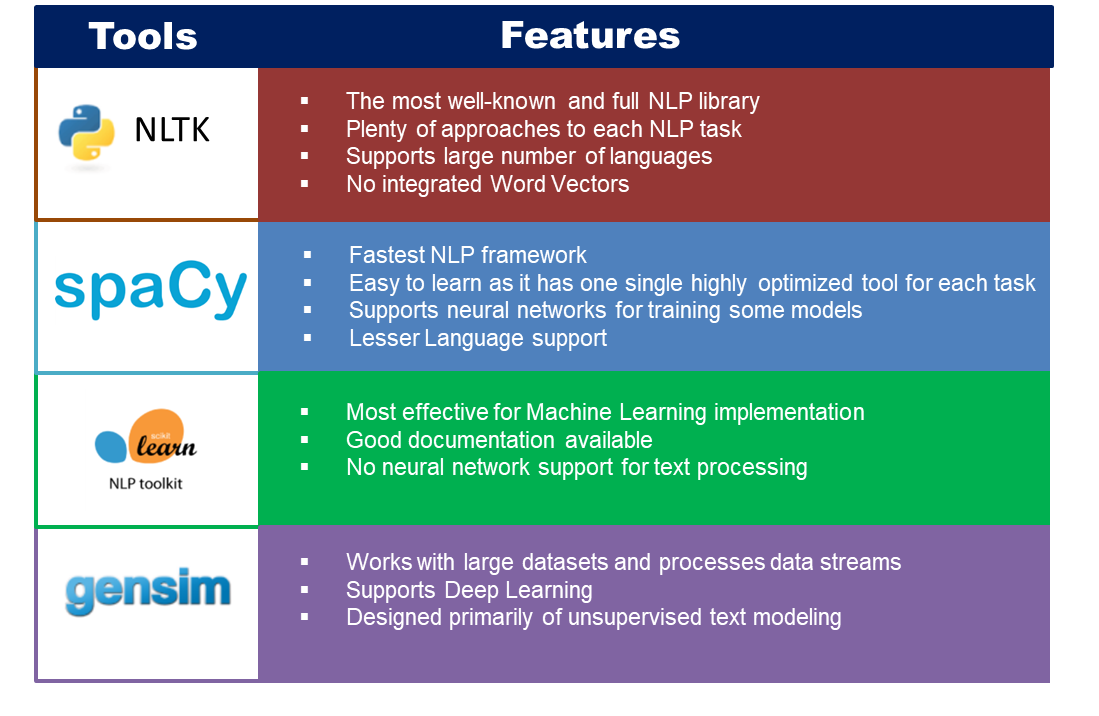
is a collection of words and/or phrases along with associated information, such as part-of-speech and sense definitions. Lexical resources are secondary to texts, and are usually created and enriched with the help of texts

**Lexical Analysis**

Lexical analysis is the process of trying to understand what words mean, intuit their context, and note the relationship of one word to others. It is often the entry point to many NLP data pipelines. Lexical analysis can come in many forms and varieties

|  |  |
| --- | --- |
| **Open a source file** | takes a source code file |
| **Convert into lower case** |  |
| **Removing special charachters** |  |
| **preserve multiple words together as an "n-gram"** |  |
| **Stop words** |  |
| **Tokenisation** | breaks down the lines of code to a series of "tokens" |
| **look up words in a dictionary and attempt to extract their meanings.** | For a compiler, this would involve **finding keywords and associating operations or variables with the tokens**.  In other contexts, such as a **chat bot**, the lookup may involve using a database to match intent.  As noted above, there are often multiple meanings for a specific word, which means that the computer has to decide what meaning the word has in relation to the sentence in which it is used. |
| **Stemming And Lemmatization** |  |

**Libraries used for processing Text**



**STOP WORDS**

The words which are generally filtered out before processing a natural language are called stop words. These are actually the most common words in any language (like articles, prepositions, pronouns, conjunctions, etc) and does not add much information to the text. Examples of a few stop words in English are “the”, “a”, “an”, “so”, “what”

Removal of stop words definitely reduces the dataset size and thus reduces the training time due to the fewer number of tokens involved in the training.

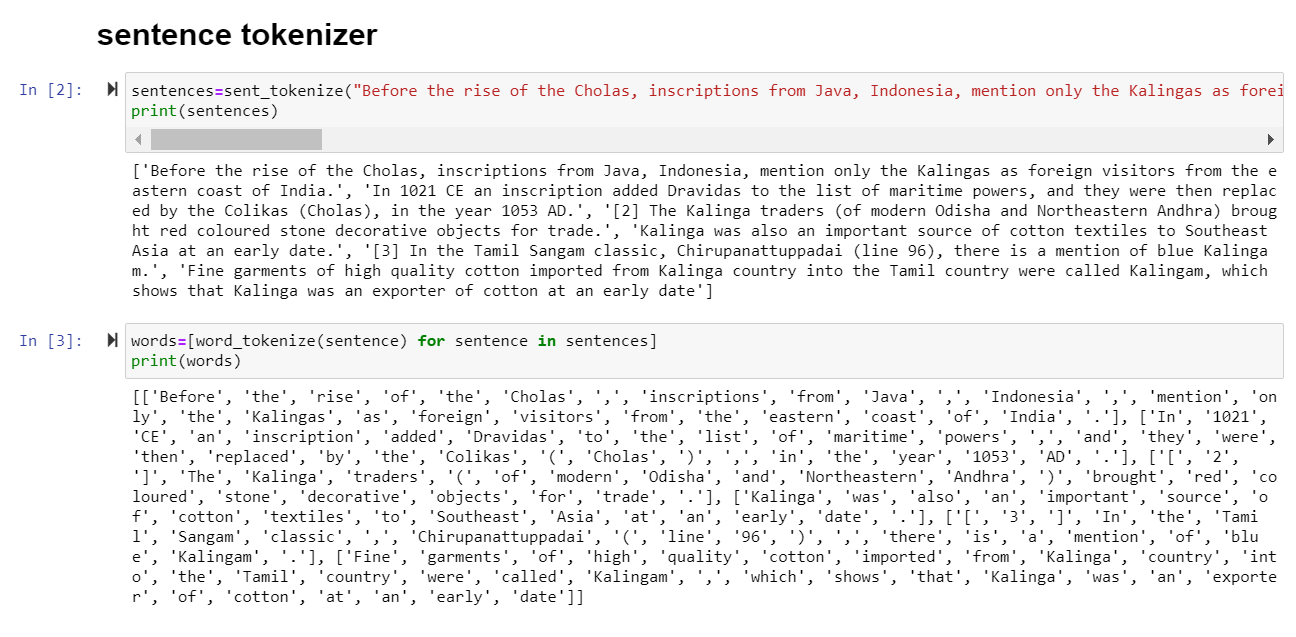
|  |  |
| --- | --- |
| **Removing stop words** | **Not removing stop words** |
| We do not always remove the stop words. The removal of stop words is highly dependent on the task we are performing and the goal we want to achieve. For example, if we are training a model that can perform the sentiment analysis task, we might not remove the stop words.  Movie review: “The movie was not good at all.” Text after removal of stop words: “movie good” | Tasks like text classification do not generally need stop words as the other words present in the dataset are more important and give the general idea of the text. So, we generally remove stop words in such tasks |

**Libraries for stopwords:**

|  |  |
| --- | --- |
| **NLTK** | **spaCy** |
| NLTK provides a plethora of algorithms to choose from for a particular problem which is boon for a researcher but a bane for a developer. | spaCy keeps the best algorithm for a problem in its toolkit and keep it updated as state of the art improves. |
| supports various languages | have statistical models for 7 languages (English, German, Spanish, French, Portuguese, Italian, and Dutch). It also supports named entities for multi language. |
| string processing library. It takes strings as input and returns strings or lists of strings as output. | ses object-oriented approach. When we parse a text, spaCy returns document object whose words and sentences are objects themselves |
| NLTK does not have support for word vectors | spaCy has **support for word vectors** |
| in **sentence tokenization**, NLTK outperforms spaCy | in **word tokenization and POS-tagging** spaCy performs better |

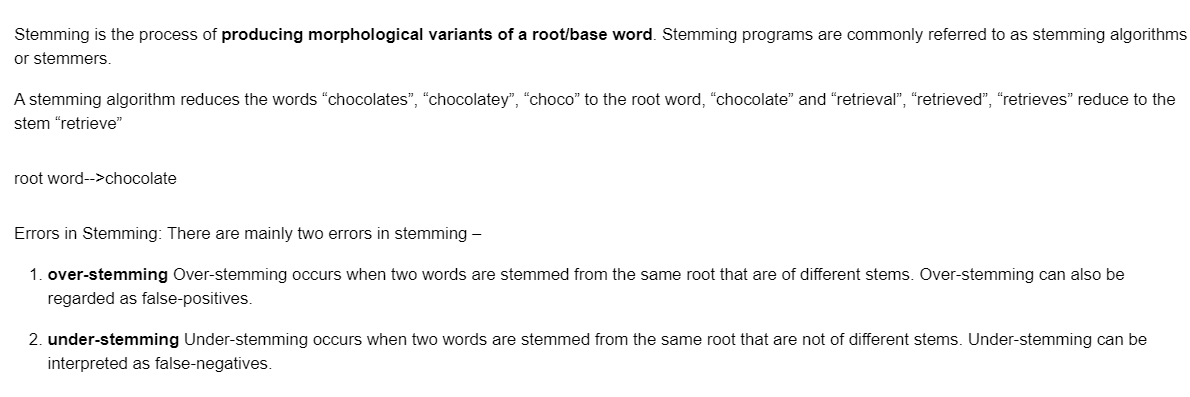
|  |  |
| --- | --- |
| **Gensim** | **Scikit-Learn** |
| **Gensim (Generate Similar)** is an **open-source software library** that uses modern statistical machine learning. According to Wikipedia, **Gensim is designed to handle large text collections using data streaming and incremental online algorithms**, which differentiates it from most other machine learning software packages that target only inmemory processing. | Scikit-Learn **needs no introduction**. It is a **free software machine learning library for Python**. It is probably the most powerful library for machine learning |

**Tokenisation**





**Stemming**



|  |  |
| --- | --- |
| Root word | Stems |
| “chocolate” | “chocolates”, “chocolatey”, “choco” |
| “retrieve” | “retrieval”, “retrieved”, “retrieves” |
| “like” | "likes", "liked", "likely", "liking" |

**Stemming algorithms:**

|  |  |  |
| --- | --- | --- |
| **Porter’s Stemmer algorithm** | It is based on the idea that the suffixes in the English language are made up of a combination of smaller and simpler suffixes. The main applications of Porter Stemmer include data mining and Information retrieval. However, its applications are only limited to English words. Also, the group of stems is mapped on to the same stem and the output stem is not necessarily a meaningful word | Example: EED -> EE means “if the word has at least one vowel and consonant plus EED ending, change the ending to EE” as ‘agreed’ becomes ‘agree’. |
| **Lovins Stemmer** | removes the longest suffix from a word then the word is recorded to convert this stem into valid words. | sitting -> sitt -> sit |
| **Dawson Stemmer** | extension of Lovins stemmer in which suffixes are stored in the reversed order indexed by their length and last letter. |  |
| **Krovetz Stemmer** | Following are the steps:  1) Convert the plural form of a word to its singular form.  2) Convert the past tense of a word to its present tense and remove the suffix ‘ing’. | ‘children’ -> ‘child’ |
| **Xerox Stemmer** |  | ‘children’ -> ‘child’  ‘understood’ -> ‘understand’  ‘whom’ -> ‘who’  ‘best’ -> ‘good’ |
| **N-Gram Stemmer** | An n-gram is a set of n consecutive characters extracted from a word in which similar words will have a high proportion of n-grams in common. | ‘INTRODUCTIONS’ for n=2 becomes : \*I, IN, NT, TR, RO, OD, DU, UC, CT, TI, IO, ON, NS, S\* |
| **Snowball Stemmer** | can map non-English words too. Since it supports other languages the Snowball Stemmers can be called a multi-lingual stemmer. The Snowball stemmers are also imported from the nltk package. This stemmer is based on a programming language called ‘Snowball’ that processes small strings and is the most widely used stemmer.  The Snowball stemmer is way more aggressive than Porter Stemmer and is also referred to as Porter2 Stemmer. Because of the improvements added when compared to the Porter Stemmer, the Snowball stemmer is having greater computational speed. |  |
| **Lancaster Stemmer** | The Lancaster stemmers are more aggressive and dynamic compared to the other two stemmers. The stemmer is really faster, but the algorithm is really confusing when dealing with small words. But they are not as efficient as Snowball Stemmers. The Lancaster stemmers save the rules externally and basically uses an iterative algorithm. |  |

**LEMMATIZATION**

