**1. What are Corpora?**

**Ans:**

A corpus is a large and structured set of machine-readable texts that have been produced in a natural communicative setting. Its plural is **corpora**. They can be derived in different ways like text that was originally electronic, transcripts of spoken language and optical character recognition, etc.

In natural language processing, a corpus contains text and speech data that can be used to train AI and machine learning systems. If a user has a specific problem or objective they want to address, they’ll need a collection of data that supports, or at least is a representation of, what they’re looking to achieve with machine learning and NLP.

**Features of a good corpus**

Large corpus size: Generally, the larger the size of a corpus, the better. Large quantities of specialized datasets are vital to training algorithms designed to perform sentiment analysis.

High-quality data: High quality is crucial when it comes to the data within a corpus. Due to the large volume of data required for a corpus, even minuscule errors in the training data can lead to large-scale errors in the machine learning system’s output.

Clean data: Data cleansing is also vital for creating and maintaining a high-quality corpus. Data cleansing allows identifying and eliminating any errors or duplicate data to create a more reliable corpus for NLP.

Balance: A high-quality corpus is a balanced corpus. While it can be tempting to fill a corpus with everything and anything available, if one doesn’t streamline and structure the data collection process, it could unbalance the relevance of the dataset.

**2. What are Tokens?**

**Ans:**

Tokens are the building blocks of Natural Language. Tokenization is a way of separating a piece of text into smaller units called tokens. Here, tokens can be either words, characters, or subwords. Hence, tokenization can be broadly classified into 3 types – word, character, and subword (n-gram characters) tokenization.

For example, consider the sentence: “Never give up”.

The most common way of forming tokens is based on space. Assuming space as a delimiter, the tokenization of the sentence results in 3 tokens – Never-give-up. As each token is a word, it becomes an example of Word tokenization.

Similarly, tokens can be either characters or subwords. For example, let us consider “smarter”:

Character tokens: s-m-a-r-t-e-r

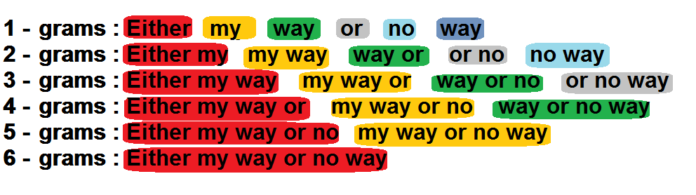
Subword tokens: smart-er

**3. What are Unigrams, Bigrams and Trigrams?**

**Ans:**

In Natural Language Processing n-gram is a contiguous sequence of n items generated from a given sample of text where the items can be characters or words and n can be any numbers like 1, 2, 3, etc.

An n-gram of size 1 is referred to as a **unigram** size 2 is a **bigram** size 3 is a **trigram**. When **N>3** this is usually referred to as four grams or five grams and so on.



**4. How to generate n-grams from text?**

I/P:

**from** nltk.util **import** ngrams, everygrams

**def** ngram\_convertor(sentence,n**=**3):

ngram\_sentence **=** ngrams(sentence**.**split(), n)

**for** item **in** ngram\_sentence:

print(item,end**=**',')

print()

sentence **=** "Life is either a daring adventure or nothing at all"

print('-'**\***25,'Unigram','-'**\***25)

ngram\_convertor(sentence,1)

print('-'**\***25,'Bigram','-'**\***25)

ngram\_convertor(sentence,2)

print('-'**\***25,'Trigram','-'**\***25)

ngram\_convertor(sentence,3)

print('-'**\***25,'Everygram','-'**\***25)

print(list(everygrams(sentence**.**split())))

O/P:

------------------------- Unigram -------------------------

('Life',),('is',),('either',),('a',),('daring',),('adventure',),('or',),('nothing',),('at',),('all',),

------------------------- Bigram -------------------------

('Life', 'is'),('is', 'either'),('either', 'a'),('a', 'daring'),('daring', 'adventure'),('adventure', 'or'),('or', 'nothing'),('nothing', 'at'),('at', 'all'),

------------------------- Trigram -------------------------

('Life', 'is', 'either'),('is', 'either', 'a'),('either', 'a', 'daring'),('a', 'daring', 'adventure'),('daring', 'adventure', 'or'),('adventure', 'or', 'nothing'),('or', 'nothing', 'at'),('nothing', 'at', 'all'),

------------------------- Everygram -------------------------

[('Life',), ('Life', 'is'), ('Life', 'is', 'either'), ('Life', 'is', 'either', 'a'), ('Life', 'is', 'either', 'a', 'daring'), ('Life', 'is', 'either', 'a', 'daring', 'adventure'), ('Life', 'is', 'either', 'a', 'daring', 'adventure', 'or'), ('Life', 'is', 'either', 'a', 'daring', 'adventure', 'or', 'nothing'), ('Life', 'is', 'either', 'a', 'daring', 'adventure', 'or', 'nothing', 'at'), ('Life', 'is', 'either', 'a', 'daring', 'adventure', 'or', 'nothing', 'at', 'all'), ('is',), ('is', 'either'), ('is', 'either', 'a'), ('is', 'either', 'a', 'daring'), ('is', 'either', 'a', 'daring', 'adventure'), ('is', 'either', 'a', 'daring', 'adventure', 'or'), ('is', 'either', 'a', 'daring', 'adventure', 'or', 'nothing'), ('is', 'either', 'a', 'daring', 'adventure', 'or', 'nothing', 'at'), ('is', 'either', 'a', 'daring', 'adventure', 'or', 'nothing', 'at', 'all'), ('either',), ('either', 'a'), ('either', 'a', 'daring'), ('either', 'a', 'daring', 'adventure'), ('either', 'a', 'daring', 'adventure', 'or'), ('either', 'a', 'daring', 'adventure', 'or', 'nothing'), ('either', 'a', 'daring', 'adventure', 'or', 'nothing', 'at'), ('either', 'a', 'daring', 'adventure', 'or', 'nothing', 'at', 'all'), ('a',), ('a', 'daring'), ('a', 'daring', 'adventure'), ('a', 'daring', 'adventure', 'or'), ('a', 'daring', 'adventure', 'or', 'nothing'), ('a', 'daring', 'adventure', 'or', 'nothing', 'at'), ('a', 'daring', 'adventure', 'or', 'nothing', 'at', 'all'), ('daring',), ('daring', 'adventure'), ('daring', 'adventure', 'or'), ('daring', 'adventure', 'or', 'nothing'), ('daring', 'adventure', 'or', 'nothing', 'at'), ('daring', 'adventure', 'or', 'nothing', 'at', 'all'), ('adventure',), ('adventure', 'or'), ('adventure', 'or', 'nothing'), ('adventure', 'or', 'nothing', 'at'), ('adventure', 'or', 'nothing', 'at', 'all'), ('or',), ('or', 'nothing'), ('or', 'nothing', 'at'), ('or', 'nothing', 'at', 'all'), ('nothing',), ('nothing', 'at'), ('nothing', 'at', 'all'), ('at',), ('at', 'all'), ('all',)]

**5. Explain Lemmatization?**

**Ans:**

Lemmatization is the grouping together of different forms of the same word. In search queries, lemmatization allows end users to query any version of a base word and get relevant results. Because search engine algorithms use lemmatization, the user is free to query any inflectional form of a word and get relevant results. For example, if the user queries the plural form of a word (routers), the search engine knows to also return relevant content that uses the singular form of the same word (router).

Lemmatization is an important aspect of natural language understanding ([NLU](https://www.techtarget.com/searchenterpriseai/definition/natural-language-understanding-NLU)) and natural language processing ([NLP](https://www.techtarget.com/searchbusinessanalytics/definition/natural-language-processing-NLP)) and plays an important role in [big data analytics](https://www.techtarget.com/searchbusinessanalytics/definition/big-data-analytics) and artificial intelligence ([AI](https://www.techtarget.com/searchenterpriseai/definition/AI-Artificial-Intelligence)). Complex algorithms use the rules of linguistic morphology, in context with a particular language's vocabulary, to group words used in speech and writing by inflected forms. [Deep learning](https://www.techtarget.com/searchenterpriseai/definition/deep-learning-deep-neural-network)is used to analyze and understand the grouping as a whole, so when any inflectional form of a word is mentioned, the base term's entire lemmatization is included.

**6. Explain Stemming?**

**Ans:**

Stemming is a technique used to extract the base form of the words by removing affixes from them. It is just like cutting down the branches of a tree to its stems. For example, the stem of the words eating, eats, eaten is eat.

Search engines use stemming for indexing the words. That’s why rather than storing all forms of a word, a search engine can store only the stems. In this way, stemming reduces the size of the index and increases retrieval accuracy.

**7. Explain Part-of-speech (POS) tagging?**

**Ans:**

Part-of-speech (POS) tagging may be defined as the process of converting a sentence in the form of a list of words, into a list of tuples. Here, the tuples are in the form of (word, tag). We can also call POS tagging a process of assigning one of the parts of speech to the given word.

In simple words, we can say that POS tagging is a task of labelling each word in a sentence with its appropriate part of speech. We already know that parts of speech include nouns, verb, adverbs, adjectives, pronouns, conjunction and their sub-categories.

Most of the POS tagging falls under **Rule Base POS tagging**, **Stochastic POS tagging** and **Transformation based tagging**.

**8. Explain Chunking or shallow parsing?**

**Ans:**

Chunking is somewhere between part of speech (POS) tagging and full language parsing, hence the name shallow parsing. If chunkers are an in-between stage then why are they relevant? The answer comes down to utility and speed.

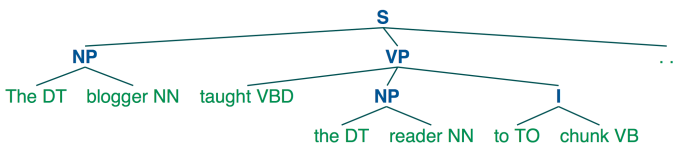
POS tagging is very fast but often doesn’t provide a ton of utility for information extraction. It’s helpful to know the POS tags, but when we try to derive information about our text we’re still swimming within the unstructured soup of words in a sentence. Knowing that word 1, 4 and 7 in our sentence are nouns won’t often won’t prove useful enough to help us reliably gain knowledge about what our sentence is actually saying; there’s too much room for mistake.

**POS Tags:**



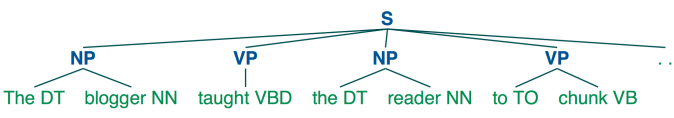
On the other hand, full parsing is extremely useful: we’re able to understand the syntactic relationship details between the words in our text, and information extraction becomes much easier to define. However, full parsing takes a very long time and will often give you information you don’t necessarily need. Some degree of parsing helps structure our text, but knowing that the determiner in the middle of our sentence is four branches down from the root and part of a nested prepositional clause within a NP clause within the main VP clause…might be overkill, as is the other parse tree produced for our sentence because the syntactic ambiguity in the prepositional clause lends itself to two interpretations: the subject in pajamas shooting an elephant, or the subject shooting the elephant that is wearing his pajamas.

**(Sort of correct) Parsing:**



Chunking is the happy middle ground that gives you enough information about the syntactic structure to reliably extract meaning from language without burdening your system with unnecessary information.

**Chunking:**



**9. Explain Noun Phrase (NP) chunking?**

**Ans:**

Text chunking is dividing sentences into non-overlapping phrases. Noun phrase chunking deals with extracting the noun phrases from a sentence. While NP chunking is much simpler than parsing, it is still a challenging task to build an accurate and very efficient NP chunker. The importance of NP chunking derives from the fact that it is used in many applications.

#### Applications:

Noun phrases can be used as a pre-processing tool before parsing the text. Due to the high ambiguity of the natural language exact parsing of the text may become very complex. In these cases chunking can be used as a pre-processing tool to partially resolve these ambiguities.

Noun phrases can be used in Information Retrieval systems. In this application the chunking can be used to retrieve the data's from the documents depending on the chunks rather than the words. In particular nouns and noun phrases are more useful for retrieval and extraction purposes.

Most of the recent work on machine translation use texts in two languages (parallel corpora) to derive useful transfer patterns. Noun phrases also have applications in aligning of text in parallel corpora. The sentences in the parallel corpora can be aligned by using the chunk information and by relating the chunks in the source and the target language. This can be done lot more easily than doing word alignment between the texts of the two languages.

Further noun phrases that are chunked can also be used in other applications where in depth parsing of the data is not necessary.

**10. Explain Named Entity Recognition?**

**Ans:**

Named entity recognition (NER) — sometimes referred to as entity chunking, extraction, or identification — is the task of identifying and categorizing key information (entities) in text. An entity can be any word or series of words that consistently refers to the same thing. Every detected entity is classified into a predetermined category. For example, an NER machine learning (ML) model might detect the word “super.AI” in a text and classify it as a “Company”.

NER is a form of natural language processing (NLP), a subfield of artificial intelligence. NLP is concerned with computers processing and analyzing natural language, i.e., any language that has developed naturally, rather than artificially, such as with computer coding languages.