1. What is Corpora?

Ans:

plural noun: corpora, A plural of Corpus

Corpus a collection of written or spoken texts.

A *corpus* is a collection of linguistic data (usually contained in a computer database) used for research, scholarship, and teaching. Also called a *text corpus*.

A *corpus* is a collection of linguistic data (usually contained in a computer database) used for research, scholarship, and teaching. Also called a *text corpus*.

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.

challenges regarding creating a corpus:

* Deciding the type of data needed to solve the problem statement
* Availability of data
* Quality of the data
* Adequacy of the data in terms of the amount

1. What are Tokens?

Ans:

Tokens" are usually individual words .

In NLP Tokens can be words, characters, or even sub-words depending on what splitting algorithm is used. Tokens can be individual words, phrases or even whole sentences.

The tokens usually become the input for the processes like parsing and text mining.

These tokens are very useful for finding patterns as well as are considered as a base step for stemming and lemmatization.

Technique used to generate tokens:

Tokenization is the process of breaking up a given text into units called tokens.

1. White space tokenization:

Perhaps this is one of the simplest techniques to tokenize a sentence or paragraph into words. In this technique, the input sentence is broken apart every time a white-space is encountered. Although this is a fast and easy way to implement tokenization, this technique only works in languages where meaningful units are separated by spaces e.g English. But for words such as living room, full moon, real estate, coffee table, this method might work accurately.

1. Dictionary based tokenization:

This is a much more advanced method than white space tokenizer. We find tokens from the sentences that are already in the dictionary. This approach needs specific guidance if the tokens in the sentence aren’t in the dictionary For languages without spaces between words, there is an additional step of word segmentation where we find sequences of characters that have a certain meaning.

1. Subword Tokenization:

This is a collection of approaches usually using unsupervised [machine learning](https://www.mygreatlearning.com/blog/what-is-machine-learning/) techniques. This method finds short sequences of characters that are often used together and assigns each of them to be a separate token. As this is an unsupervised method, sometimes we may encounter tokens that have actually no real meaning.

1. What are Unigrams, Bigrams, Trigrams?

Ans:

N-Grams are phrases cut out of a sentence with a number of consecutive words. Thus a Unigram takes a sentence and gives us all the words that we fence. A Bigram takes a sentence and gives us sets of two consecutive words in the sentence. A Trigram gives sets of three consecutive words in a sentence.

Let me explain with an example.

Unigram - [Let] [me] [explain] [with] [an] [example.]

Bigram [let me] [me explain] [explain with] [with an] [an example]

Trigram [let me explain] [me explain with] [explain with an] [with an example]

In NLP it is a An N-gram is a sequence of N tokens (or words).

These are statistical Language Models to learn the probability distribution words.

1. How to generate n-grams from text?

Ans:

Steps:

* Split the sentence/ text to tokens/words
* Loop the split words in the sequence to combine the words(N) to form a N-grams

Ex in python:

| 1  2  3  4  5  6  7  8  9  10 | # Creating a function to generate N-Grams  def generate\_ngrams(text, WordsToCombine):  words = text.split()  output = []  for i in range(len(words)- WordsToCombine+1):  output.append(words[i:i+WordsToCombine])  return output    # Calling the function  generate\_ngrams(text='this is a very good book to study', WordsToCombine=3) |
| --- | --- |

Using NLTK library

# NLTK function to generate ngrams

import nltk

from nltk.util import ngrams

samplText='this is a very good book to study'

NGRAMS=ngrams(sequence=nltk.word\_tokenize(samplText), n=3)

for grams in NGRAMS:

print(grams)

1. Explain Lemmatization

Ans:

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemmatization.

Lemmatisation (or lemmatization) in linguistics is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form

In [computational linguistics](https://en.wikipedia.org/wiki/Computational_linguistics), **lemmatisation** is the algorithmic process of determining the [l**emma**](https://en.wikipedia.org/wiki/Lemma_(morphology))of a word based on its intended meaning.

A **lemma** (plural lemmas or lemmata) is the canonical form,[[1]](https://en.wikipedia.org/wiki/Lemma_(morphology)#cite_note-1) dictionary form, or citation form of a set of [word](https://en.wikipedia.org/wiki/Word) forms.[[2]](https://en.wikipedia.org/wiki/Lemma_(morphology)#cite_note-2) In [English](https://en.wikipedia.org/wiki/English_language), for example, break, breaks, broke, broken and breaking are forms of the same [lexeme](https://en.wikipedia.org/wiki/Lexeme), with break as the lemma by which they are indexed.

**Inflection** (or [inflexion](https://en.wiktionary.org/wiki/inflexion#English)) is a process of [word formation](https://en.wikipedia.org/wiki/Word_formation)[[1]](https://en.wikipedia.org/wiki/Inflection#cite_note-1) in which a word is modified to express different [grammatical categories](https://en.wikipedia.org/wiki/Grammatical_category) such as [tense](https://en.wikipedia.org/wiki/Grammatical_tense), [case](https://en.wikipedia.org/wiki/Grammatical_case), [voice](https://en.wikipedia.org/wiki/Grammatical_voice), [aspect](https://en.wikipedia.org/wiki/Grammatical_aspect), [person](https://en.wikipedia.org/wiki/Grammatical_person), [number](https://en.wikipedia.org/wiki/Grammatical_number), [gender](https://en.wikipedia.org/wiki/Grammatical_gender), [mood](https://en.wikipedia.org/wiki/Grammatical_mood), [animacy](https://en.wikipedia.org/wiki/Animacy), and [definiteness](https://en.wikipedia.org/wiki/Definiteness)

1. Explain Stemming

Ans:

Stemming is the process of reducing a word to its stem that affixes to suffixes and prefixes or to the roots of words known as "lemmas"

Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers.

Example: “chocolates”, “chocolatey”, “choco” to the root word, “chocolate” and “retrieval”, “retrieved”, “retrieves” reduce to the stem “retrieve”.

**Stemming** is an important part of the pipelining process in Natural language processing. The input to the stemmer is tokenized words.

**Stemming is a natural language processing technique that is used to reduce words to their base form, also known as the root form.** The process of stemming is used to normalize text and make it easier to process. It is an important step in text pre-processing, and it is commonly used in information retrieval and text mining applications.

There are several different algorithms for stemming:

* Porter stemmer
* The Porter stemmer is the most widely used algorithm, and it is based on a set of heuristics that are used to remove common suffixes from words.
* Snowball stemmer
* The Snowball stemmer is a more advanced algorithm that is based on the Porter stemmer, but it also supports several other languages in addition to English.
* Lancaster stemmer.
* The Lancaster stemmer is a more aggressive stemmer and it is less accurate than the Porter stemmer and Snowball stemmer.

Uses:

Stemming can be useful for several natural language processing tasks such as text classification, information retrieval, and text summarization.

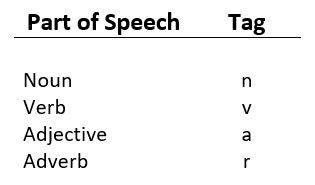
Cons:

However, stemming can also have some negative effects such as reducing the readability of the text, and it may not always produce the correct root form of a word.

1. Explain Part-of-speech (POS) tagging

Ans:

In corpus linguistics, part-of-speech tagging (POS tagging or PoS tagging or POST), also called grammatical tagging **is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition and its context**.

a process of converting a sentence to forms – list of words, list of tuples (where each tuple is having a form *(word, tag)*). The tag in case of is a part-of-speech tag, and signifies whether the word is a noun, adjective, verb, and so on. 

**Default tagging** is a basic step for the part-of-speech tagging

* It is performed using the DefaultTagger class. The DefaultTagger class takes ‘tag’ as a single argument. **NN** is the tag for a singular noun. DefaultTagger is most useful when it gets to work with the most common part-of-speech tag. That's why a noun tag is recommended.

Example:

| # Loading Libraries  from nltk.tag import DefaultTagger    # Defining Tag  tagging = DefaultTagger('NN')    # Tagging  tagging.tag(['Hello', 'Geeks']) |
| --- |

**Output :**

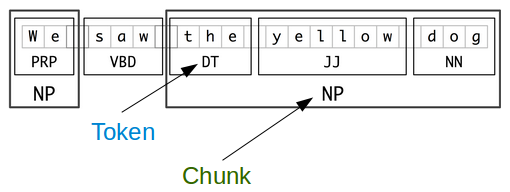
[('Hello', 'NN'), ('Geeks', 'NN')]

1. Explain Chunking or shallow parsing

Ans:

Shallow parsing (also chunking or light parsing) is an analysis of a sentence which first identifies constituent parts of sentences (nouns, verbs, adjectives, etc.) and then links them to higher order units that have discrete grammatical meanings (noun groups or phrases, verb groups, etc.).

However, it does not specify their internal structure, nor their role in the main sentence.



"The smaller boxes show the word-level tokenization and part-of-speech tagging, while the large boxes show higher-level chunking. Each of these larger boxes is called a chunk. Like tokenization, which omits whitespace, chunking usually selects a subset of the tokens. Also like tokenization, the pieces produced by a chunker do not overlap in the source text.

Example:

import nltk

if \_\_name\_\_ == "\_\_main\_\_":

sentence = "We saw the yellow dog"

tokens = nltk.word\_tokenize(sentence)

print("tokens = %s") %(tokens)

tagged = nltk.pos\_tag(tokens)

entities = nltk.chunk.ne\_chunk(tagged)

print("entities = %s") %(entities)

Output:

tokens = ['We', 'saw', 'the', 'yellow', 'dog']

entities = (S We/PRP saw/VBD the/DT yellow/JJ dog/NN)

1. Explain Noun Phrase (NP) chunking

Ans:

**Noun Chunking** or **Noun Phrase Chunking**, or **Base Noun Phrases** [1](https://www.dataknowsall.com/nounphrase.html#fn:SPACY). Chunking builds upon these grammatical parts to identify groups of words that go together to form symbolic meaning. This can be an adjective that goes along with a noun or a group of nouns related to each other. Below is a simple example using Spacy to demonstrate how it works.

Example:

import spacy

nlp = spacy.load("en\_core\_web\_sm")

doc = nlp("Five Words in Orange Neon")

for chunk in doc.noun\_chunks:

print(chunk.text)

Five Words

Orange Neon

In this simple five-word sentence, we can see how it identified **Five Words** and **Orange Neon** as the chunks.

1. Explain Named Entity Recognition

Ans:

The named entity recognition (NER) is one of the most popular data preprocessing task.

It involves the identification of key information in the text and classification into a set of predefined categories. An entity is basically the thing that is consistently talked about or refer to in the text.

NER is the form of NLP.

At its core, NLP is just a two-step process, below are the two steps that are involved:

* Detecting the entities from the text
* Classifying them into different categories

**Methods of NER**

* One way is to train the model for multi-class classification using different machine learning algorithms, but it requires a lot of labelling. In addition to labelling the model also requires a deep understanding of context to deal with the ambiguity of the sentences. This makes it a challenging task for a simple machine learning algorithm.
* Another way is that Conditional random field that is implemented by both NLP Speech Tagger and NLTK. It is a probabilistic model that can be used to model sequential data such as words. The CRF can capture a deep understanding of the context of the sentence. In this model, the input 



* **Deep Learning Based NER:** deep learning NER is much more accurate than previous method, as it is capable to assemble words. This is due to the fact that it used a method called word embedding, that is capable of understanding the semantic and syntactic relationship between various words. It is also able to learn analyzes topic-specific as well as high level words automatically. This makes deep learning NER applicable for performing multiple tasks. Deep learning can do most of the repetitive work itself, hence researchers for example can use their time more efficiently.

**Implementation**

* In this implementation, we will perform Named Entity Recognition using two different frameworks: Spacy and NLTK. This code can be run on colab, however for visualization purpose. recommend the local environment. We can install the following frameworks using pip install
* First, we performed Named Entity recognition using Spacy.
* Python3

# command to run before code

! pip install spacy

! pip install nltk

! python -m spacy download en\_core\_web\_sm

# imports and load spacy english language package

import spacy

from spacy import displacy

from spacy import tokenizer

nlp = spacy.load('en\_core\_web\_sm')

#Load the text and process it

# I copied the text from python wiki

text =("Python is an interpreted, high-level and general-purpose programming language

"Pythons design philosophy emphasizes code readability with"

"its notable use of significant indentation."

"Its language constructs and object-oriented approach aim to"

"help programmers write clear and"

"logical code for small and large-scale projects")

# text2 = # copy the paragraphs from <https://www.python.org/doc/essays/>

doc = nlp(text)

#doc2 = nlp(text2)

sentences = list(doc.sents)

print(sentences)

# tokenization

for token in doc:

print(token.text)

# print entities

ents = [(e.text, e.start\_char, e.end\_char, e.label\_) for e in doc.ents]

print(ents)

# now we use displaycy function on doc2

displacy.render(doc, style='ent', jupyter=True)

* Below is a list and their meaning of spacy entity tags:



* Now we performed the named entity recognition task on NLTK.
* Python3

| # import modules and download packages  import nltk  nltk.download('words')  nltk.download('punkt')  nltk.download('maxent\_ne\_chunker')  nltk.download('averaged\_perceptron\_tagger')  nltk.download('state\_union')  from nltk.corpus import state\_union  from nltk.tokenize import PunktSentenceTokenizer    # process the text and print Named entities  # tokenization  train\_text = state\_union.raw()    sample\_text = state\_union.raw("2006-GWBush.txt")  custom\_sent\_tokenizer = PunktSentenceTokenizer(train\_text)  tokenized = custom\_sent\_tokenizer.tokenize(sample\_text)  # function  def get\_named \_entity():  try:  for i in tokenized:  words = nltk.word\_tokenize(i)  tagged = nltk.pos\_tag(words)  namedEnt = nltk.ne\_chunk(tagged, binary=False)  namedEnt.draw()  except:  pass  get\_named\_entity() |
| --- |