Natural Language Processing (NLP) with Python

**Table of Contents:**

1. [What is Natural Language Processing?](https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0#7d22)
2. [Applications of NLP](https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0#7458)
3. [Understanding Natural Language Processing (NLP)](https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0#e9b8)
4. [Rule-based NLP vs. Statistical NLP](https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0#aab0)
5. [Components of Natural Language Processing (NLP)](https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0#1bcf)
6. [Current challenges in NLP](https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0#c0d8)
7. [Easy to Use NLP Libraries](https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0#62b7)
8. [Exploring Features of NLTK](https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0#7ec0)
9. [Word Cloud](https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0#2847)
10. [Stemming](https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0#f0db)
11. [Lemmatization](https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0#3be2)
12. [Part-of-Speech (PoS) tagging](https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0#32ff)
13. [Chunking](https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0#b706)
14. [Chinking](https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0#01f7)
15. [Named Entity Recognition (NER)](https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0#a03e)
16. [WordNet](https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0#6724)
17. [Bag of Words](https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0#2143)
18. [TF-IDF](https://medium.com/towards-artificial-intelligence/natural-language-processing-nlp-with-python-tutorial-for-beginners-1f54e610a1a0#27f3)

**What is Natural Language Processing?**

Computers and machines are great at working with tabular data or spreadsheets. However, as human beings generally communicate in words and sentences, not in the form of tables. Much information that humans speak or write is unstructured. So it is not very clear for computers to interpret such. In natural language processing (NLP), the goal is to make computers understand the unstructured text and retrieve meaningful pieces of information from it. Natural language Processing (NLP) is a subfield of [**artificial intelligence**](https://mld.ai/mldcmu), in which its depth involves the interactions between computers and humans.

**Applications of NLP:**

* Machine Translation.
* Speech Recognition.
* Sentiment Analysis.
* Question Answering.
* Summarization of Text.
* Chatbot.
* Intelligent Systems.
* Text Classifications.
* Character Recognition.
* Spell Checking.
* Spam Detection.
* Autocomplete.
* Named Entity Recognition.
* Predictive Typing.

**Understanding Natural Language Processing (NLP):**

We, as humans, perform natural language processing (NLP) considerably well, but even then, we are not perfect. We often misunderstand one thing for another, and we often interpret the same sentences or words differently.

For instance, consider the following sentence, we will try to understand its interpretation in many different ways:

**Example 1:**

Figure 2: NLP Example Sentence with text: “I saw a man on a hill with a telescope.”

Figure 2: NLP example sentence with the text: “I saw a man on a hill with a telescope.”

These are some interpretations of the sentence shown above.

* There is a man on the hill, and I watched him with my telescope.
* There is a man on the hill, and he has a telescope.
* I’m on a hill, and I saw a man using my telescope.
* I’m on a hill, and I saw a man who has a telescope.
* There is a man on a hill, and I saw him something with my telescope.

**Example 2:**

Figure 3: NLP example sentence with the text: “Can you help me with the can?”

Figure 3: NLP example sentence with the text: “Can you help me with the can?”

In the sentence above, we can see that there are two “can” words, but both of them have different meanings. Here the first “can” word is used for question formation. The second “can” word at the end of the sentence is used to represent a container that holds food or liquid.

Hence, from the examples above, we can see that language processing is not “deterministic” (the same language has the same interpretations), and something suitable to one person might not be suitable to another. Therefore, Natural Language Processing (NLP) has a non-deterministic approach. In other words, Natural Language Processing can be used to create a new intelligent system that can understand how humans understand and interpret language in different situations.

**Rule-based NLP vs. Statistical NLP:**

Natural Language Processing is separated in two different approaches:

**Rule-based Natural Language Processing:**

It uses common sense reasoning for processing tasks. For instance, the freezing temperature can lead to death, or hot coffee can burn people’s skin, along with other common sense reasoning tasks. However, this process can take much time, and it requires manual effort.

**Statistical Natural Language Processing:**

It uses large amounts of data and tries to derive conclusions from it. Statistical NLP uses machine learning algorithms to train NLP models. After successful training on large amounts of data, the trained model will have positive outcomes with deduction.

**Comparison:**

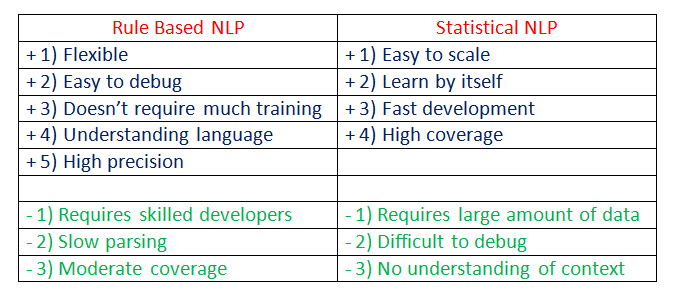


Figure 4: Rule-Based NLP vs. Statistical NLP.

**Components of Natural Language Processing (NLP):**

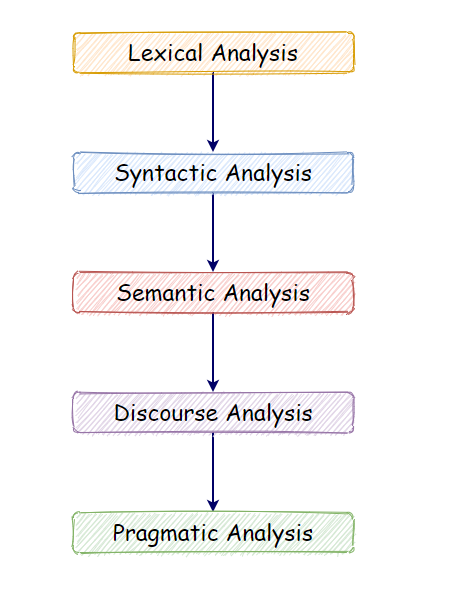


Figure 5: Components of Natural Language Processing (NLP).

**a. Lexical Analysis:**

With lexical analysis, we divide a whole chunk of text into paragraphs, sentences, and words. It involves identifying and analyzing words’ structure.

**b. Syntactic Analysis:**

Syntactic analysis involves the analysis of words in a sentence for grammar and arranging words in a manner that shows the relationship among the words. For instance, the sentence “The shop goes to the house” does not pass.

**c. Semantic Analysis:**

Semantic analysis draws the exact meaning for the words, and it analyzes the text meaningfulness. Sentences such as “hot ice-cream” do not pass.

**d. Disclosure Integration:**

Disclosure integration takes into account the context of the text. It considers the meaning of the sentence before it ends. For example: “He works at Google.” In this sentence, “he” must be referenced in the sentence before it.

**e. Pragmatic Analysis:**

Pragmatic analysis deals with overall communication and interpretation of language. It deals with deriving meaningful use of language in various situations.

**Current challenges in NLP:**

1. Breaking sentences into tokens.
2. Tagging parts of speech (POS).
3. Building an appropriate vocabulary.
4. Linking the components of a created vocabulary.
5. Understanding the context.
6. Extracting semantic meaning.
7. Named Entity Recognition (NER).
8. Transforming unstructured data into structured data.
9. Ambiguity in speech.

**Easy to use NLP libraries:**

**a.**[**NLTK (Natural Language Toolkit)**](https://www.nltk.org/)**:**

The NLTK Python framework is generally used as an education and research tool. It’s not usually used on production applications. However, it can be used to build exciting programs due to its ease of use.

**Features:**

* Tokenization.
* Part Of Speech tagging (POS).
* Named Entity Recognition (NER).
* Classification.
* Sentiment analysis.
* Packages of chatbots.

**Use-cases:**

* Recommendation systems.
* Sentiment analysis.
* Building chatbots.

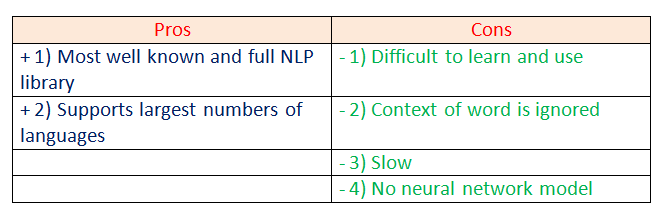


Figure 6: Pros and cons of using the NLTK framework.

**b. [spaCy](https://spacy.io/" \t "_blank):**

spaCy is an open-source natural language processing Python library designed to be fast and production-ready. spaCy focuses on providing software for production usage.

**Features:**

* Tokenization.
* Part Of Speech tagging (POS).
* Named Entity Recognition (NER).
* Classification.
* Sentiment analysis.
* Dependency parsing.
* Word vectors.

**Use-cases:**

* Autocomplete and autocorrect.
* Analyzing reviews.
* Summarization.

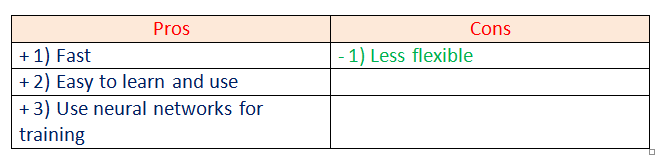


Figure 7: Pros and cons of the spaCy framework.

**c. [Gensim](https://pypi.org/project/gensim/" \t "_blank):**

Gensim is an NLP Python framework generally used in topic modeling and similarity detection. It is not a general-purpose NLP library, but it handles tasks assigned to it very well.

**Features:**

* Latent semantic analysis.
* Non-negative matrix factorization.
* TF-IDF.

**Use-cases:**

* Converting documents to vectors.
* Finding text similarity.
* Text summarization.

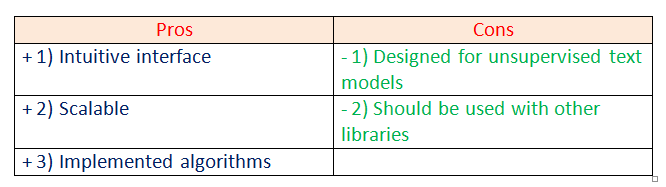


Figure 8: Pros and cons of the Gensim framework.

**d.**[**Pattern**](https://github.com/clips/pattern)**:**

Pattern is an NLP Python framework with straightforward syntax. It’s a powerful tool for scientific and non-scientific tasks. It is highly valuable to students.

**Features:**

* Tokenization.
* Part of Speech tagging.
* Named entity recognition.
* Parsing.
* Sentiment analysis.

**Use-cases:**

* Spelling correction.
* Search engine optimization.
* Sentiment analysis.

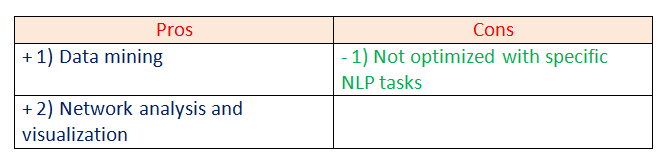


Figure 9: Pros and cons of the Pattern framework.

**e. [TextBlob](https://textblob.readthedocs.io/en/dev/" \t "_blank):**

TextBlob is a Python library designed for processing textual data.

**Features:**

* Part-of-Speech tagging.
* Noun phrase extraction.
* Sentiment analysis.
* Classification.
* Language translation.
* Parsing.
* Wordnet integration.

**Use-cases:**

* Sentiment Analysis.
* Spelling Correction.
* Translation and Language Detection.

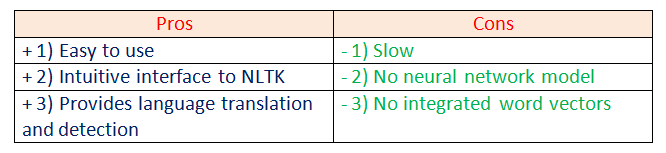


Figure 10: Pros and cons of the TextBlob library.

For this tutorial, we are going to focus more on the NLTK library. Let’s dig deeper into natural language processing by making some examples.

**Exploring Features of NLTK:**

**a. Open the text file for processing:**

First, we are going to open and read the file which we want to analyze.

#open the text file

text\_file = open("T1.txt")

text = text\_file.read()

print(type(text))

print("\n")

print(text)

print(len(text))

**b. Import required libraries:**

For various data processing cases in NLP, we need to import some libraries. In this case, we are going to use NLTK for Natural Language Processing. We will use it to perform various operations on the text.

#import required libraries

import nltk

from nltk import sent\_tokenize

from nltk import word\_tokenize

**c. Sentence tokenizing:**

By tokenizing the text with sent\_tokenize( ), we can get the text as sentences.

# tokenize the text by sentences

sentences = sent\_tokenize(text)

#how many sentences are there?

print(len(sentences))

#print the sentences

print(sentences)

In the example above, we can see the entire text of our data is represented as sentences and also notice that the total number of sentences here is **9**.

**d. Word tokenizing:**

By tokenizing the text with word\_tokenize( ), we can get the text as words.

words = word\_tokenize(text)

print(len(words))

print(words)

Next, we can see the entire text of our data is represented as words and also notice that the total number of words here is **144**.

**e. Find the frequency distribution:**

Let’s find out the frequency of words in our text.

#import required libraries

from nltk.probability import FreqDist

#Find the frequency

fdist = FreqDist(words)

#print 10 most common words

fdist.most\_common(10)

Notice that the most used words are punctuation marks and stopwords. We will have to remove such words to analyze the actual text.

**f. Plot the frequency graph:**

Let’s plot a graph to visualize the word distribution in our text.

import matplotlib.pyplot as ply

fdist.plot(10)

In the graph above, notice that a period “.” is used nine times in our text. Analytically speaking, punctuation marks are not that important for natural language processing. Therefore, in the next step, we will be removing such punctuation marks.

**g. Remove punctuation marks:**

Next, we are going to remove the punctuation marks as they are not very useful for us. We are going to use isalpha( ) method to separate the punctuation marks from the actual text. Also, we are going to make a new list called words\_no\_punc, which will store the words in lower case but exclude the punctuation marks.

words\_no\_punc =[]

#removing punctuation marks

for w in words:

if w.isalpha():

words\_no\_punc.append(w.lower())

#print the words without punctuation marks

print(words\_no\_punc)

print("\n")

#length

print(len(words\_no\_punc))

As shown above, all the punctuation marks from our text are excluded. These can also cross-check with the number of words.

**h. Plotting graph without punctuation marks:**

fdist = FreqDist(words\_no\_punc)

fdist.most\_common(10)

#plot the most common words on graph

fdist.plot(10)

Notice that we still have many words that are not very useful in the analysis of our text file sample, such as “and,” “but,” “so,” and others. Next, we need to remove coordinating conjunctions.

**i. List of stopwords:**

from nltk.corpus import stopwords

#list of stopwords

stopwords = stopwords.words("english")

print(stopwords)

**j. Removing stopwords:**

# Empty list to store clean words

clean\_words =[]

for w in words\_no\_punc:

if w not in stopwords:

clean\_words.append(w)

print(clean\_words)

print("\n")

print(len(clean\_words))

**k. Final frequency distribution:**

#Frequency distribution

fdist = FreqDist(clean\_words)

fdist.most\_common(10)

fdist.plot(10) #plot the most common words on graph

As shown above, the final graph has many useful words that help us understand what our sample data is about, showing how essential it is to perform data cleaning on NLP.

Next, we will cover various topics in NLP with coding examples.

**Word Cloud:**

Word Cloud is a data visualization technique. In which words from a given text display on the main chart. In this technique, more frequent or essential words display in a larger and bolder font, while less frequent or essential words display in smaller or thinner fonts. It is a beneficial technique in NLP that gives us a glance at what text should be analyzed.

**Properties:**

1. **font\_path**: It specifies the path for the fonts we want to use.
2. **width**: It specifies the width of the canvas.
3. **height**: It specifies the height of the canvas.
4. **min\_font\_size**: It specifies the smallest font size to use.
5. **max\_font\_size:** It specifies the largest font size to use.
6. **font\_step**: It specifies the step size for the font.
7. **max\_words**: It specifies the maximum number of words on the word cloud.
8. **stopwords**: Our program will eliminate these words.
9. **background\_color:** It specifies the background color for canvas.
10. **normalize\_plurals**: It removes the trailing “s” from words.

**Read the full documentation on [WordCloud](https://amueller.github.io/word_cloud/generated/wordcloud.WordCloud.html" \t "_blank).**

**Word Cloud Python Implementation:**

#Library to form wordcloud

from wordcloud import WordCloud

#library to plot the wordcloud

import matplotlib.pyplot as plt

#generating the wordcloud

wordcloud = WordCloud().generate(text)

#plot the wordcloud

plt.figure(figsize =(12,12))

plt.imshow(wordcloud)

#to remove thw axis value

plt.axis("off")

plt.show()

As shown in the graph above, the most frequent words display in larger fonts. The word cloud can be displayed in any shape or image.

For instance: In this case, we are going to use the following circle image, but we can use any shape or any image.

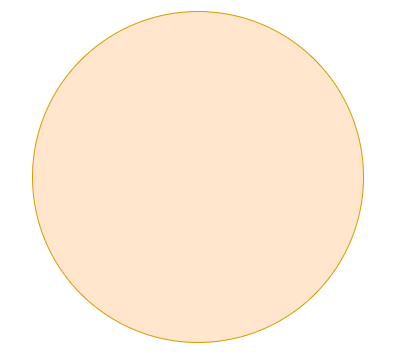


Figure 33: Circle image shape for our word cloud.

As shown above, the word cloud is in the shape of a circle. As we mentioned before, we can use any shape or image to form a word cloud.

**Word CloudAdvantages:**

* They are fast.
* They are engaging.
* They are simple to understand.
* They are casual and visually appealing.

**Word Cloud Disadvantages:**

* They are non-perfect for non-clean data.
* They lack the context of words.

**Stemming:**

We use Stemming to normalize words. In English and many other languages, a single word can take multiple forms depending upon context used. For instance, the verb “study” can take many forms like “studies,” “studying,” “studied,” and others, depending on its context. When we tokenize words, an interpreter considers these input words as different words even though their underlying meaning is the same. Moreover, as we know that NLP is about analyzing the meaning of content, to resolve this problem, we use stemming.

Stemming normalizes the word by truncating the word to its stem word. For example, the words “studies,” “studied,” “studying” will be reduced to **“studi,”** making all these word forms to refer to only one token. Notice that stemming may not give us a dictionary, grammatical word for a particular set of words.

Let’s take an example:

**a. Porter’s Stemmer Example 1:**

In the code snippet below, we show that all the words truncate to their stem words. However, notice that the stemmed word is not a dictionary word.

#stemming Example

#import stemming Library

from nltk.stem import PorterStemmer

porter = PorterStemmer()

# word-list for stemming

word\_list =["Study","Studying","Studies","Studied"]

for w in word\_list:

print(porter.stem(w))

**b. Porter’s Stemmer Example 2:**

In the code snippet below, many of the words after stemming did not end up being a recognizable dictionary word.

#stemming Example

#import stemming Library

from nltk.stem import PorterStemmer

porter = PorterStemmer()

# word-list for stemming

word\_list =["Studies","leaves","decreases","plays"]

for w in word\_list:

print(porter.stem(w))

**c. SnowballStemmer:**

SnowballStemmer generates the same output as porter stemmer, but it supports many more languages.

#stemming Example

#import stemming Library

from nltk.stem import SnowballStemmer

snowball = SnowballStemmer("english")

# word-list for stemming

word\_list =["Study","Studying","decreases","plays"]

for w in word\_list:

print(snowball.stem(w))

**d. Languages supported by snowball stemmer:**

#stemming Example

#import stemming Library

from nltk.stem import SnowballStemmer

#print languages supported

SnowballStemmer.languages

**Various Stemming Algorithms:**

**a. Porter’s Stemmer:**

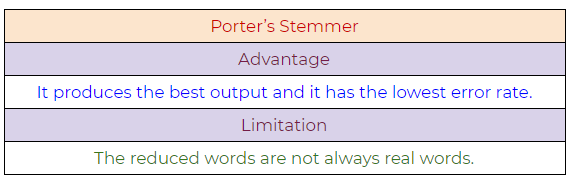


Figure 40: Porter’s Stemmer NLP algorithm, pros, and cons.

**b. Lovin’s Stemmer:**

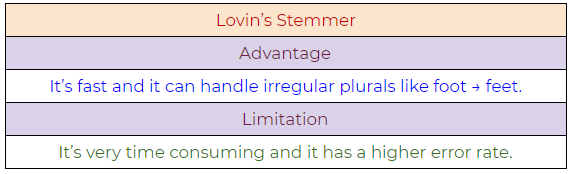


Figure 41: Lovin’s Stemmer NLP algorithm, pros, and cons.

**c. Dawson’s Stemmer:**

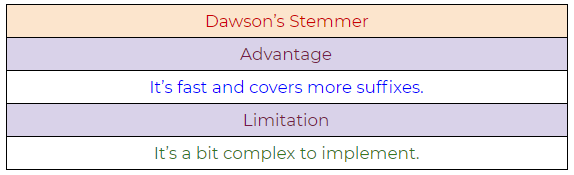


Figure 42: Dawson’s Stemmer NLP algorithm, pros, and cons.

**d. Krovetz Stemmer:**

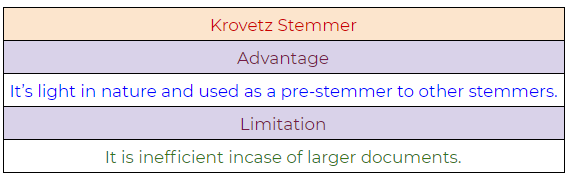


Figure 43: Krovetz Stemmer NLP algorithm, pros, and cons.

**e. Xerox Stemmer:**

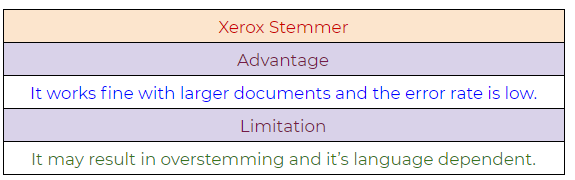


Figure 44: Xerox Stemmer NLP algorithm, pros, and cons.

**f. Snowball Stemmer:**

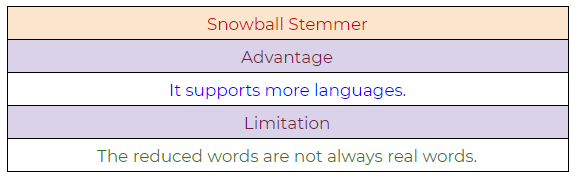


Figure 45: Snowball Stemmer NLP algorithm, pros, and cons.

**Lemmatization:**

Lemmatization tries to achieve a similar base “stem” for a word. However, what makes it different is that it finds the dictionary word instead of truncating the original word. Stemming does not consider the context of the word. That is why it generates results faster, but it is less accurate than lemmatization.

If accuracy is not the project’s final goal, then stemming is an appropriate approach. If higher accuracy is crucial and the project is not on a tight deadline, then the best option is amortization (Lemmatization has a lower processing speed, compared to stemming).

Lemmatization takes into account Part Of Speech (POS) values. Also, lemmatization may generate different outputs for different values of POS. We generally have four choices for POS:

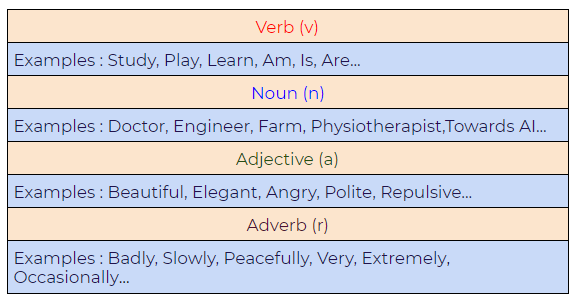


Figure 46: Part of Speech (POS) values in lemmatization.

**Difference between Stemmer and Lemmatizer:**

**a. Stemming:**

Notice how on stemming, the word “studies” gets truncated to “studi.”

#stemming Example

#import stemming Library

from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

print(stemmer.stem('studies'))

**b. Lemmatizing:**

During lemmatization, the word “studies” displays its dictionary word “study.”

#stemming Example

#import stemming Library

from nltk.stem import WordNetLemmatizer

lem = WordNetLemmatizer()

print(lem.lemmatize('studies'))

**Python Implementation:**

**a. A basic example demonstrating how a lemmatizer works**

In the following example, we are taking the PoS tag as “verb,” and when we apply the lemmatization rules, it gives us dictionary words instead of truncating the original word:

#stemming Example

#import stemming Library

from nltk.stem import WordNetLemmatizer

lem = WordNetLemmatizer()

word\_list =['study','studying','studies','studied']

for w in word\_list:

print(lem.lemmatize(w,pos='v'))

Figure 49: Simple lemmatization example with the NLTK framework.

**b. Lemmatizer with default PoS value**

The default value of PoS in lemmatization is a noun(n). In the following example, we can see that it’s generating dictionary words:

#stemming Example

#import stemming Library

from nltk.stem import WordNetLemmatizer

lem = WordNetLemmatizer()

word\_list =['studies','leaves','decreases','plays']

for w in word\_list:

print(lem.lemmatize(w))

**c. Another example demonstrating the power of lemmatizer**

#stemming Example

#import stemming Library

from nltk.stem import WordNetLemmatizer

lem = WordNetLemmatizer()

word\_list =["am","is","are","was","were"]

for w in word\_list:

print(lem.lemmatize(w, pos ="v"))

output will be ‘be’

**d. Lemmatizer with different POS values**

#stemming Example

#import stemming Library

from nltk.stem import WordNetLemmatizer

lem = WordNetLemmatizer()

print(lem.lemmatize("studying", pos ="v"))

print(lem.lemmatize("studying", pos ="n"))

print(lem.lemmatize("studying", pos ="a"))

print(lem.lemmatize("studying", pos ="r"))

**Part of Speech Tagging (PoS tagging):**

**Why do we need Part of Speech (POS)?**

Figure 53: Sentence example, “can you help me with the can?”

Figure 53: Sentence example, “can you help me with the can?”

Parts of speech (PoS) tagging is crucial for syntactic and semantic analysis. Therefore, for something like the sentence above, the word “can” has several semantic meanings. The first “can” is used for question formation. The second “can” at the end of the sentence is used to represent a container. The first “can” is a verb, and the second “can” is a noun. Giving the word a specific meaning allows the program to handle it correctly in both semantic and syntactic analysis.

Below, please find a list of Part of Speech (PoS) tags with their respective examples:

**1. CC: Coordinating Conjunction**

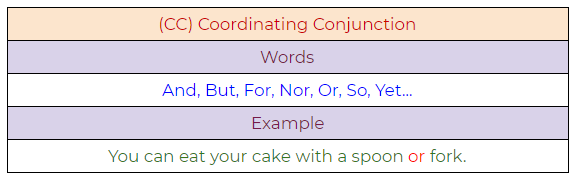


Figure 54: Coordinating conjunction example.

**2. CD: Cardinal Digit**

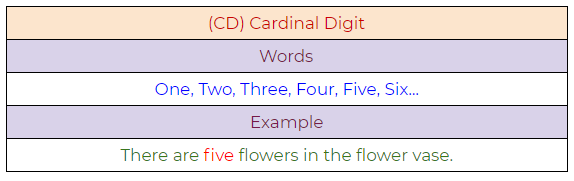


Figure 55: Cardinal digit example.

**3. DT: Determiner**

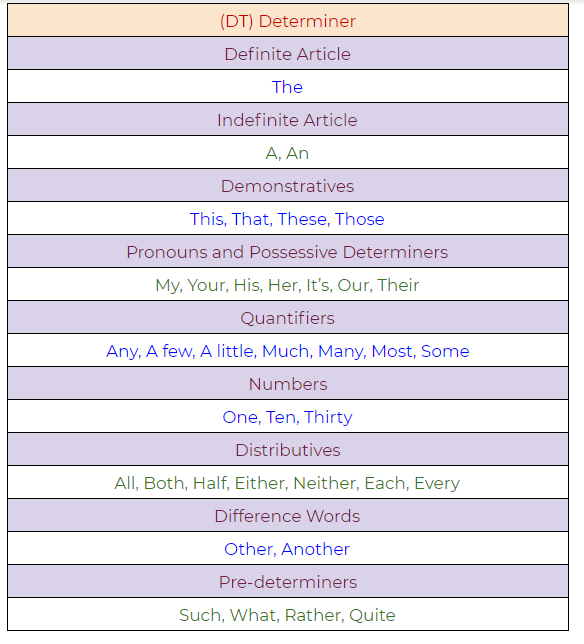


Figure 56: A determiner example.

**4. EX: Existential There**

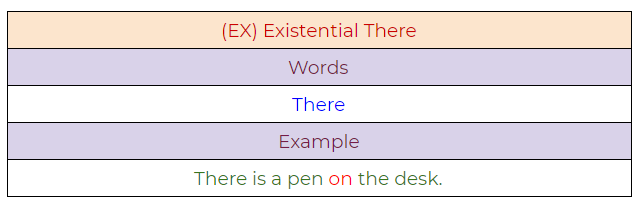


Figure 57: Existential “there” example.

**5. FW: Foreign Word**



Figure 58: Foreign word example.

**6. IN: Preposition / Subordinating Conjunction**

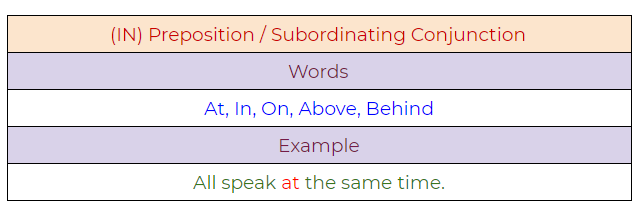


Figure 59: Preposition/Subordinating conjunction.

**7. JJ: Adjective**

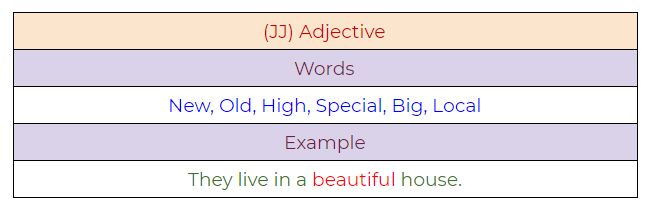


Figure 60: Adjective example.

**8. JJR: Adjective, Comparative**

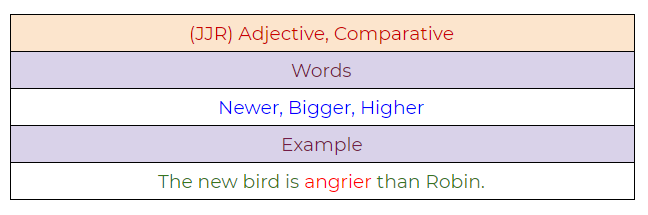


Figure 61: Adjective, comparative example.

**9. JJS: Adjective, Superlative**

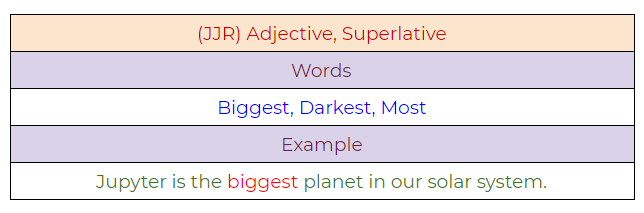


Figure 62:

**10. LS: List Marker**

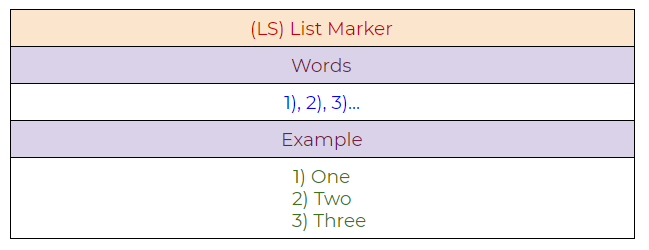


Figure 63: List marker example.

**11. MD: Modal**

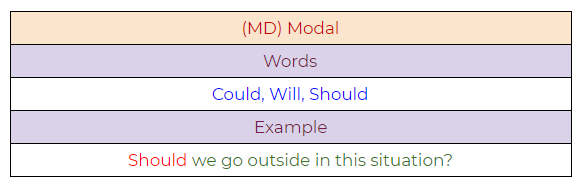


Figure 64:

**12. NN: Noun, Singular**

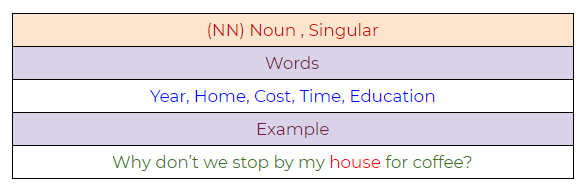


Figure 65: Noun, singular example.

**13. NNS: Noun, Plural**

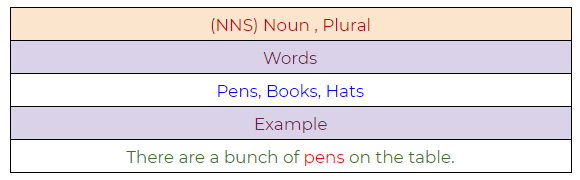


Figure 66: Noun, plural example.

**14. NNP: Proper Noun, Singular**

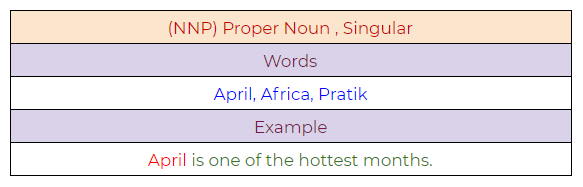


Figure 67: Proper noun, singular example.

**15. NNPS: Proper Noun, Plural**

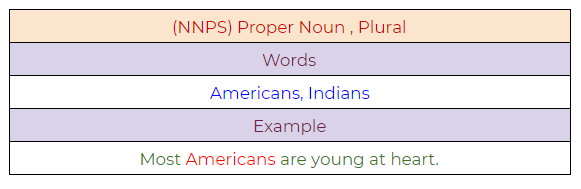


Figure 68: Proper noun, plural example.

**16. PDT: Predeterminer**

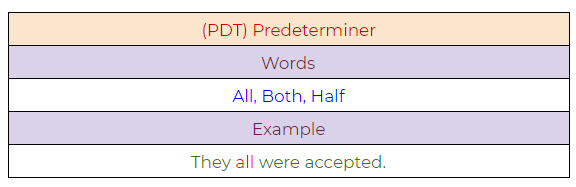


Figure 69: Predeterminer example.

**17. POS: Possessive Endings**

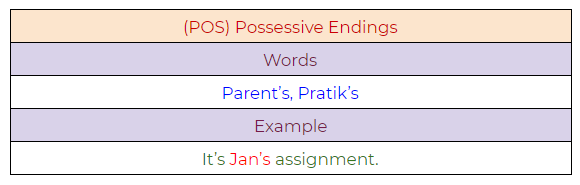


Figure 70: Possessive endings example.

**18. PRP: Personal Pronoun**

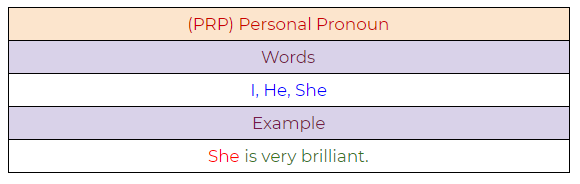


Figure 71: Personal pronoun example.

**19. PRP$: Possessive Pronoun**

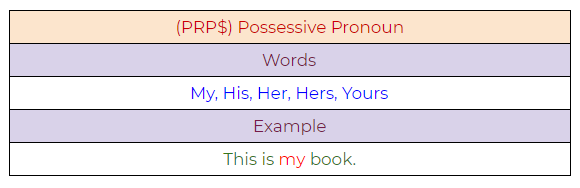


Figure 72: Possessive pronoun example.

**20. RB: Adverb**

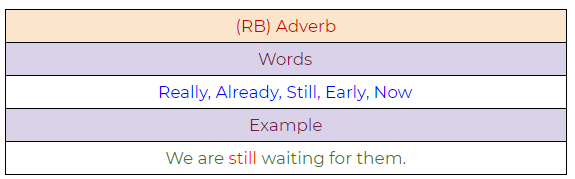


Figure 73: Adverb example.

**21. RBR: Adverb, Comparative**



Figure 74: Adverb, comparative example.

**22. RBS: Adverb, Superlative**

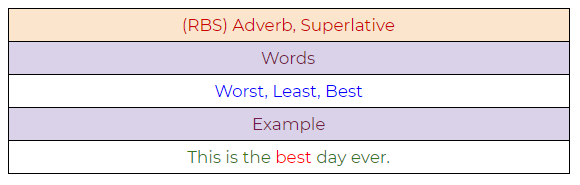


Figure 75: Adverb, superlative example.

**23. RP: Particle**

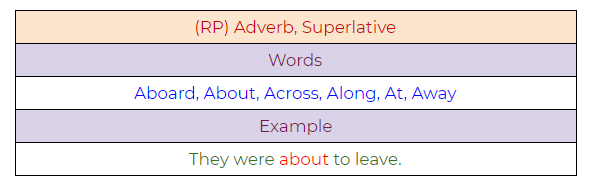


Figure 76: Particle example.

**24. TO: To**

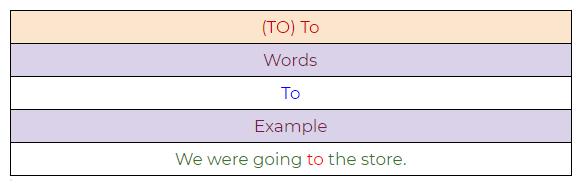


Figure 77: To example.

**25. UH: Interjection**

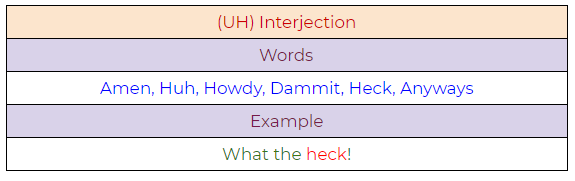


Figure 78: Interjection example.

**26. VB: Verb, Base Form**



Figure 79: Verb, base form example.

**27. VBD: Verb, Past Tense**

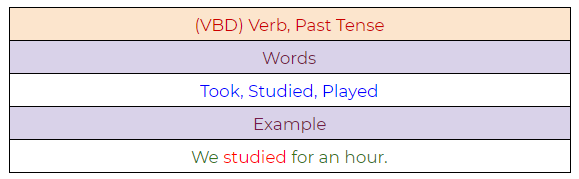


Figure 80: Verb, past tense example.

**28. VBG: Verb, Present Participle**

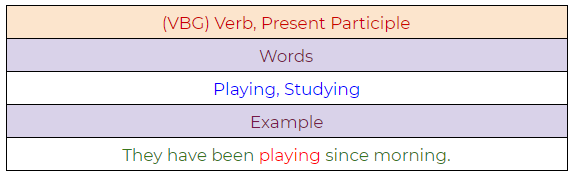


Figure 81: Verb, present participle example.

**29. VBN: Verb, Past Participle**

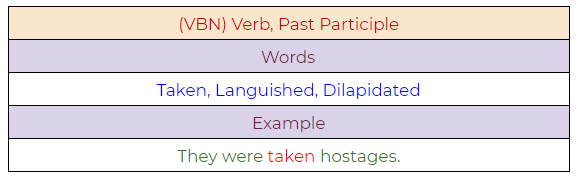


Figure 82: Verb, past participle.

**30. VBP: Verb, Present Tense, Not Third Person Singular**

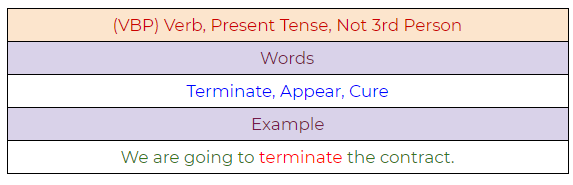


Figure 83: Verb, present tense, not third-person singular.

**31. VBZ: Verb, Present Tense, Third Person Singular**

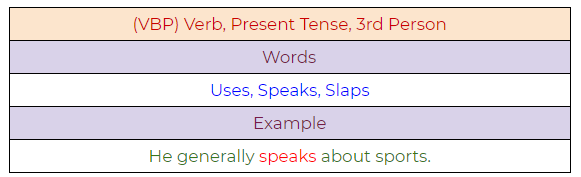


Figure 84: Verb, present tense, third-person singular.

**32. WDT: Wh — Determiner**

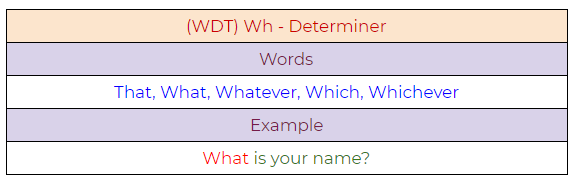


Figure 85: Determiner example.

**33. WP: Wh — Pronoun**

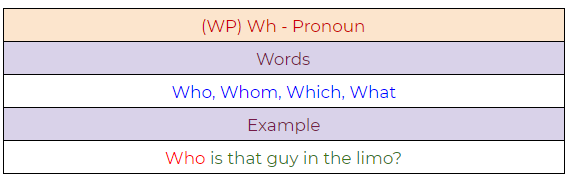


Figure 86: Pronoun example.

**34. WP$ : Possessive Wh — Pronoun**

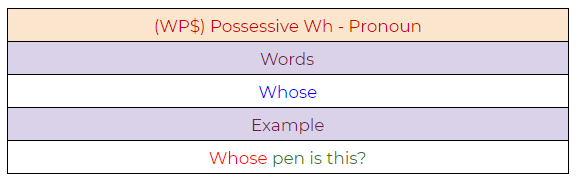


Figure 87: Possessive pronoun example.

**35. WRB: Wh — Adverb**

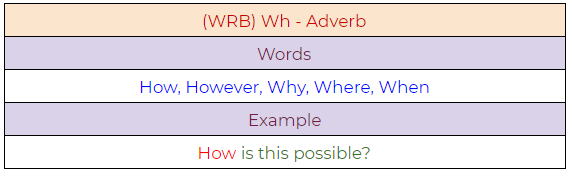


Figure 88: Adverb example.

**Python Implementation:**

**a. A simple example demonstrating PoS tagging.**

#Pos tagging

tag = nltk.pos\_tag(["Studying","Study"])

print(tag)

**b. A full example demonstrating the use of PoS tagging.**

#Pos tagging Example

sentence ="A very beutiful young lady is walking on the beach"

#Tokenizing words

token\_words = word\_tokenize(sentence)

for words in token\_words:

tagged\_words = nltk.pos\_tag(token\_words)

print(tagged\_words)

**Chunking:**

Chunking means to extract meaningful phrases from unstructured text. By tokenizing a book into words, it’s sometimes hard to infer meaningful information. It works on top of Part of Speech(PoS) tagging. Chunking takes PoS tags as input and provides chunks as output. Chunking literally means a group of words, which breaks simple text into phrases that are more meaningful than individual words.

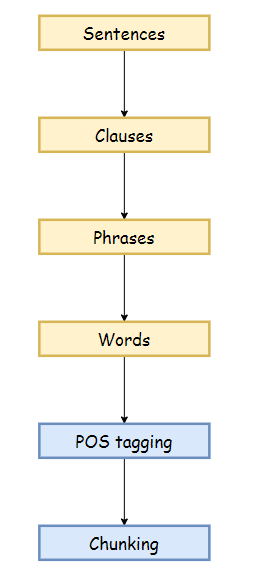


Figure 91: The chunking process in NLP.

Before working with an example, we need to know what phrases are? Meaningful groups of words are called phrases. There are five significant categories of phrases.

1. Noun Phrases (NP).
2. Verb Phrases (VP).
3. Adjective Phrases (ADJP).
4. Adverb Phrases (ADVP).
5. Prepositional Phrases (PP).

**Phrase structure rules:**

* S(Sentence) → NP VP.
* NP → {Determiner, Noun, Pronoun, Proper name}.
* VP → V (NP)(PP)(Adverb).
* PP → Pronoun (NP).
* AP → Adjective (PP).

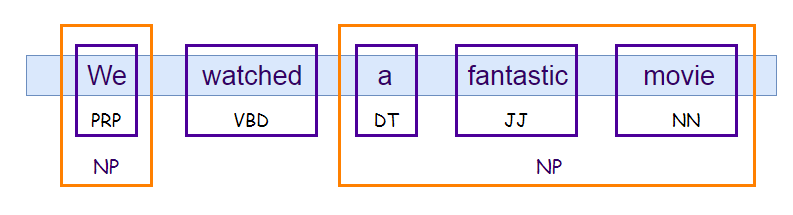
**Example:**

Figure 92: A chunking example in NLP.