



# Tokenization

Through tokenization, we can conveniently separate text by word or sentence. With this method, you can work with smaller chunks of text that remain coherent and meaningful even without the context of the whole. In our first step, we convert unstructured data into structured data, which can be analysed more easily.

When you're analysing text, you'll be tokenizing by word and tokenizing by sentence.

- **Tokenizing by word:** Words are like the atoms of natural language. The smallest unit of meaning that still makes sense on its own. When you tokenize your text by word, you can identify the words that are frequently used. Suppose you analyse a group of job ads and find that the word "Python" frequently appears. Those statistics could indicate that Python knowledge is in high demand, but you would need to dig deeper to find out more.
- When you **Tokenize by sentence**, you can analyse how the words relate to one another.

## Guides & Processes

---

[Parts Of Speech](#)

[Parts Of Speech Tagging](#)

[Types of POS Tagging](#)

[MultiLanguages POS Tagging](#)

# Parts Of Speech

The **part of speech** explains how a word is used in a sentence.

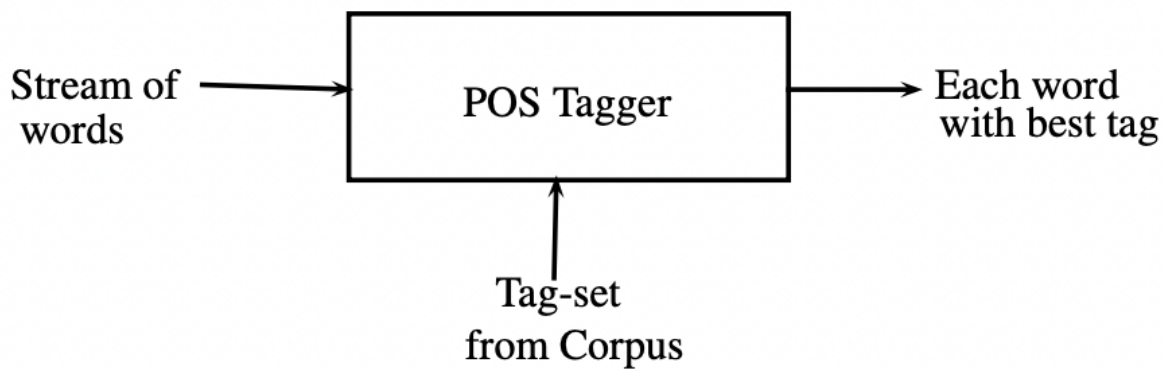
There are eight parts of speech (POS) in English: noun, verb, pronoun, preposition, adverb, conjunction, participle, and article.

The process of classifying words into their parts of speech and labeling them accordingly is known as **part-of-speech tagging** , **POS-tagging**, or **simply tagging**. Parts of speech are also known as **word classes** or **lexical categories**. They are important as they give significant amount of information about word and its neighbours. It is true for nouns and verbs. The collection of tags used for a particular task is known as a **tagset**.

# Parts Of Speech Tagging

Part of Speech Tagging (POS Tagging) refers to tagging every word in a text with a tag. The tag assignment is based on both the definition and the context of the word - i.e., how it relates to adjacent and related words within a phrase, sentence, or paragraph. It can be used to perform various natural language processing tasks by understanding syntactical components.

These are the parts of speech: nouns, verbs, adverbs, adjectives, pronouns, conjunctions, and their subcategories.



Process of Tagging

Applications:

- **Named Entity Recognition(NER)**
- **Sentiment Analysis**
- **Question Answering**
- **Word Sense Disambiguation**

POS Tagging uses NLTK library i.e., Natural Language Toolkit . It is a built in model that is trained by using **Penn Treebank POS Corpus** , **C5 tagset: 61 tags**, **C7 tagset: 146 tags** , **Brown Corpus tagset (87 tags)**.

To install NLTK Library on your local setup : `pip install nltk`

NLTK provides us with a list or set of tags, and from those options, we will provide labels to every word, run this: `nltk.help.upenn_tagset()`

Input:

```
import nltk
nltk.download('averaged_perceptron_tagger')
from nltk import word_tokenize, pos_tag
sentence = "I will move to Himachal Pradesh forever!"
tokens = word_tokenize(sentence)
tags = pos_tag(tokens)
print(tags)
```

Output:

```
[('I', 'PRP'), ('will', 'MD'), ('move', 'VB'), ('to', 'TO'), ('Himachal', 'NNP'), ('Pradesh', 'NNP'), ('forever', 'RB'), ('!', '.'), ('.', '.')]
```

As you can see from the above output, each word has a tag.

There are diagnostic rules for determining the category appropriate for a given word in context; e.g.s: if a word follows a determiner, it is a noun, e.g., “the song was a hit.” If a word precedes a noun, is not a determiner and modifies the noun’s meaning, it is an adjective

```
word_tag_pairs = nltk.bigrams(brown_news_tagged)

noun_preceders = [a[1] for (a, b) in word_tag_pairs if b[1] == 'NOUN']

fdist = nltk.FreqDist(noun_preceders)

[tag for (tag, _) in fdist.most_common()]
```

**Rules:**

- A tagset's selection depends on the application and the language used
- The input is a word sequence and the employed tagset; the output is an association between each word and its "best" tag
- A given word may have more than one tag (ambiguity).
- It is the task of the PoS tagger to resolve these ambiguities by selecting an appropriate tag based on the context of the word

Another Library used for POS Tagging : **spaCy**

**spaCy** is one of the best text analysis library. spaCy is much faster and accurate than NLTKTagger and TextBlob.

To install Spacy Library on your local setup : `pip install spacy`

## Named Entity Recognition


Named entity recognition refers to the identification of words in a sentence as an entity e.g. the name of a person, place, organization, etc.

The task of named entity recognition (NER):

- Find spans of text that constitute proper names
- Tag the type of the entity.

**spaCy** Code:

Google Colaboratory

 [https://colab.research.google.com/drive/1\\_a\\_ibK0g-\\_LJFct5ALUSgUH7Gevj4iA2?usp=sharing](https://colab.research.google.com/drive/1_a_ibK0g-_LJFct5ALUSgUH7Gevj4iA2?usp=sharing)




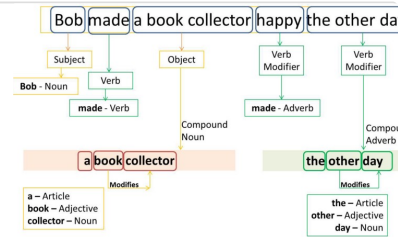
## References:

<http://krchowdhary.com/nlsp-lect/nlplect-10.pdf>

## An introduction to part-of-speech tagging and the Hidden Markov Model

by Divya Godayal An introduction to part-of-speech tagging and the Hidden Markov Model by Sachin Malhotra [<https://medium.com/@sachinmalhotra>] and Divya Godayal [<https://medium.com/@divyagodayal>] Source:

 <https://www.freecodecamp.org/news/an-introduction-to-part-of-speech-tagging-and-the-hidden-markov-model-953d45338f24/>



# Types of POS Tagging

- **Rule-based POS Tagging**



Rule-based taggers use dictionary or lexicon for getting possible tags for tagging each word. If the word has more than one possible tag, then rule-based taggers use hand-written rules to identify the correct tag.



Disambiguation can also be performed in rule-based tagging by analyzing the linguistic features of a word along with its preceding as well as following words. For example, suppose if the preceding word of a word is article then word must be a noun.

Two-stage architecture for Rule-based POS Tagging

- **First stage** – In the first stage, it uses a dictionary to assign each word a list of potential parts-of-speech.
- **Second stage** – In the second stage, it uses large lists of hand-written disambiguation rules to sort down the list to a single part-of-speech for each word.

- **Stochastic POS Tagging**



The model that includes frequency or probability (statistics) can be called stochastic. Any number of different approaches to the problem of part-of-speech tagging can be referred to as stochastic tagger.

- **Hidden Markov Model (HMM) POS Tagging**

This is a probabilistic model that uses both tag sequence probabilities and word frequency measurements.

To model any problem using a Hidden Markov Model we need a set of observations and a set of possible states. The states in an HMM are hidden. In the part of speech tagging problem, the **observations** are the words themselves in the given sequence. The **states**, which are hidden, these would be the POS tags for the words.

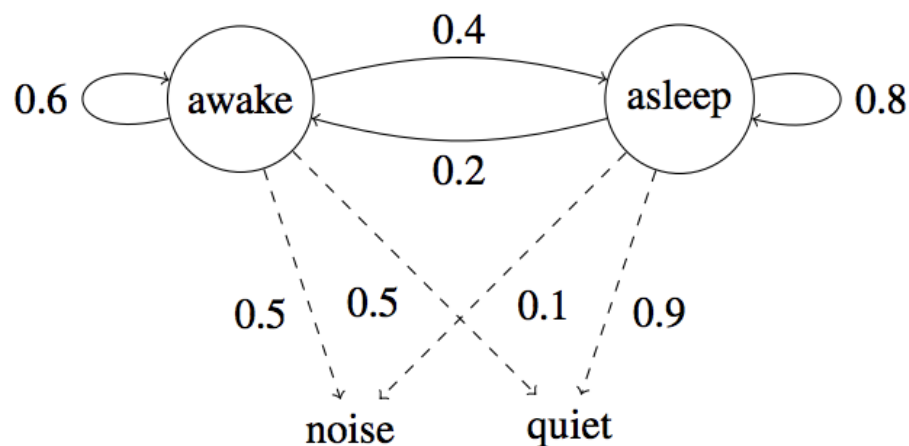
Simplified Formula:

$$P(\text{word} \mid \text{tag}) * P(\text{tag} \mid \text{previous } n \text{ tags})$$

$$P(q_1, \dots, q_n) = \prod_{i=1}^n P(q_i | q_{i-1})$$

Markovian Property for N previous Observations

- The **transition probabilities** would be somewhat like  $P(\text{VP} | \text{NP})$  that is, what is the probability of the current word having a tag of Verb Phrase given that the previous tag was a Noun Phrase.
- **Emission probabilities** would be  $P(\text{john} | \text{NP})$  or  $P(\text{will} | \text{VP})$  that is, what is the probability that the word is, say, John given that the tag is a Noun Phrase.



- One is the **emission** probabilities, which represent the probabilities of making certain observations given a particular state. For example, we have  $P(\text{noise} | \text{awake}) = 0.5$ . This is an emission probability.
- The other ones is **transition** probabilities, which represent the probability of transitioning to another state given a particular state. For example, we have  $P(\text{asleep} | \text{awake}) = 0.4$ . This is a transition probability.

We can exploit the independence assumptions in the HMM to define an efficient algorithm that returns the tag sequence with the highest probability.

- **VITERBI ALGORITHM:**

The decoding algorithm used for HMMs is called the Viterbi algorithm. First of all, we need to set up a **probability matrix called lattice** where we have **columns as our observables** (words of a sentence in the same sequence as



in sentence) & **rows as hidden states** (all possible POS Tags are known).

#### Explanation:

- **Create lattice Viterbi**  $N(\text{states/POS Tags}) \times T(\text{Observables/words})$ , also initialize back pointer ( $N \times T$ ) which will help us trace back the most probable POS Tag sequence
- **Initialize the first column of Viterbi matrix** with  $\text{initial\_probability\_distribution}(\text{1st row in 'A'}) \times \text{Emission probabilities for all tags given the 1st word (as we did for 'Janet')}$
- Set back pointers first column as 0 (representing no previous tags for the 1st word)
- Now, using a nested loop with the outer loop over all words & inner loop over all states:

**Calculate  $V(s(\text{state}), t(\text{observable})) = \max: V_{t-1} * a(s', s) * b_s(O_t)$**


here  $s'$  refers to the previous state.

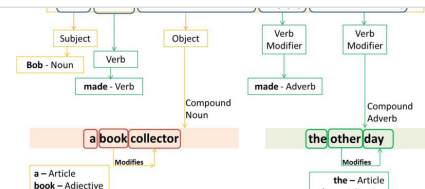
- **Set backpointer[s,t] = previous tag** from which we moved on this tag. For example. In calculating  $V_2(2)$ , we calculated the value taking NNP as a previous tag. Hence we will store **backpointer[MD,will] = NNP** & likewise.
- Once the nested loop is over, calculate **best\_path\_probability by taking max value from the last column of the Viterbi matrix** (i.e max probability for the best tag for the last word). Also, **select the tag with the highest probability for the last word as best\_path\_pointer variable**
- Now, **using this best\_path\_pointer, traceback** (using back pointer matrix) the previous state i.e **backpointer[best\_tag\_chosen, 'bill']** & recursively go back to the initial word i.e 'janet' where we set back pointer =0 will be used as terminating state.

#### References:

An introduction to part-of-speech tagging and the Hidden Markov Model


by Divya Godayal An introduction to part-of-speech tagging and the Hidden Markov Model by Sachin Malhotra [<https://medium.com/@sachinmalhotra>] and Divya Godayal [<https://medium.com/@divyagodayal>] Source:

 <https://www.freecodecamp.org/news/an-introduction-to-part-of-speech-tagging-and-the-hidden-markov-model-953d45338f24/>



POS Tagging | Part of Speech Tagging in NLP | Hidden Markov Models in NLP | Viterbi Algorithm in NLP

Code - <https://github.com/campusx-official/spacy-pos-tagging>

 [https://www.youtube.com/watch?v=269IGagoJfs&list=RDCMUCCWi3hpq\\_Pe03nGxuS7isg&start\\_radio=1&rv=269IGagoJfs&t=404](https://www.youtube.com/watch?v=269IGagoJfs&list=RDCMUCCWi3hpq_Pe03nGxuS7isg&start_radio=1&rv=269IGagoJfs&t=404)



# MultiLanguages POS Tagging

- Different languages usually define different sets of POS tags. Universal dependencies: Unify the POS tags across languages
- Amount of labeled data varies across languages. Low resource languages might benefit from high resource ones.


SpaCy Library is a Python-based open-source Natural Language Processing toolkit. It is built specifically for commercial use, allowing you to create applications that process and comprehend large amounts of text.

## Models for Human Languages:

<https://stanfordnlp.github.io/stanfordnlp/models.html>

## Examples Explained for English, German, French, Chinese, Japanese POS using SpaCy:

Google Colaboratory

 <https://colab.research.google.com/drive/1lbCe4knB-Aa1c4amDY7qE4w9qH7P5fZs#scrollTo=Eb3Q7pAFZlj2>



Languages supported by NLTK are Chinese, Danish, Dutch, English, Finnish, French, German, Hungarian, Italian, Norwegian, Portuguese, Romanian, Russian, Spanish, Swedish.

## NLP Libraries for Automatic Language Identification of Text Data In Python:

- **LangDetect**

This library is the direct port of Google's [language-detection](#) library from Java to Python and can recognize over 50 languages.

- **Spacy-langdetect**

```
def spacy_language_detection(text, model):  
  
    pipeline = list(dict(model.pipeline).keys())  
  
    if(not "language_detector" in pipeline):  
        model.add_pipe(LanguageDetector(), name = "language_detector", last=True)  
  
    doc = model(text)  
  
    return doc._.language
```

### English and French Text:

```
english_text = """Niyel, a Dakar-based company that designs, implements,  
and evaluates advocacy campaigns to change policies, behaviors, and practices,  
will support the researchers in using the results to influence the implementation of  
AI-friendly policies.  
"""  
  
french_text = """Intelligence artificielle : la solution pour améliorer l'accès au cr  
édit en Afrique ?  
Déjà une réalité au Kenya, en Afrique du Sud et au Nigeria, l'évaluation du risque cr  
édit via  
l'intelligence artificielle dispose d'un fort potentiel en Afrique de l'Ouest malgré  
les inquiétudes liées à la protection de la vie privée.""",  
  
# Load the pretrained model from spacy models' hub  
pre_trained_model = spacy.load("en_core_web_sm")  
  
# Detection on English text  
print(spacy_language_detection(english_text, pre_trained_model))  
  
# Detection on French text  
print(spacy_language_detection(french_text, pre_trained_model))
```

Output:

- **Detection on English text** shows {'language': 'en', 'score': 0.9999963977276909}, almost 100% confidence that the text is in English.
- **Detection on French text** shows {'language': 'fr', 'score': 0.9999963767662121}, almost 100% confidence that the text is in French.