
# KURUNJI VENKATRAMANA GOWDA POLYTECHNIC SULLIA-574327

## 5TH SEMESTER

## AI/ML WEEK-11

# NATURAL LANGUAGE PROCESSING

## Introduction

Computers and Machines are great while working with tabular data or Spreadsheets. However, human beings generally communicate in words and sentences, not in the form of tables or spreadsheets, and most of the information that humans speak or write is present in an unstructured manner.
So it is not very understandable for computers to interpret these languages.

Therefore, In natural language processing (NLP), our aim is to make the computer's unstructured text understandable and retrieve meaningful information from it.

**Let's define Natural Language Processing (NLP) formally,**

Natural language Processing (NLP) is a subfield of artificial intelligence, that involves the interactions between computers and humans.

So, In this article, we will discuss some of the basic concepts related to NLP.
This article is part of a blog series on Natural Language Processing (NLP).

**This is part-1 of the blog series on the Step by Step Guide to Natural Language Processing.** Important Note
After the completion of some topics, there is some practice (Test your Knowledge) questions given that you have to solve and give the answer in the comment box so that you can check your understanding of a particular topic.

### What is Natural Language Processing?

Natural Language Processing (NLP) is a subfield of Computer Science and Artificial Intelligence that deals with interactions between computers and human (natural) languages. This becomes crucial when we want to apply Machine Learning or Deep Learning Algorithms to a dataset that contains text and speech.

**For Example,** we can use NLP to create AI systems such as,

- Speech Recognition,
- Document Summarization,
- Machine Translation,
- Spam Detection,
- Named Entity Recognition,
- Question Answering,
- Autocomplete,
- Predictive Typing, etc.

In modern days, most of our smartphones have a speech recognition system. These smartphones use NLP to understand the natural language and give the response. Also, most of the peoples use laptops which operating system has built-in speech recognition.

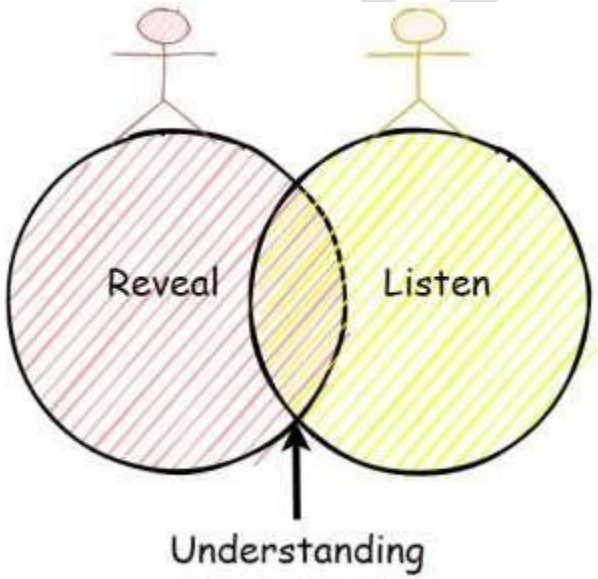
Test Your Knowledge

Which of the below options is the field of Natural Language Processing?

- Computer Science
- Artificial Intelligence
- Computational linguistics
- All of the above



Understanding Natural Language Processing



We, as humans, it's not a very difficult task to perform natural language processing (NLP) but even then, we are not perfect. We often misunderstand one thing for another and often interpret the same sentences or words in a different manner.

For instance, consider the following sentences and try to understand its interpretation in many different ways:

Example 1

**Sentence: I saw a student on a hill with a microscope.**

These are various interpretations of the above sentence which are shown below:

- There is a student on the hill, and I watched him with my microscope.
- There is a student on the hill, and he has a microscope.
- I'm on a hill, and I saw a student using my microscope.
- I'm on a hill, and I saw a student who has a microscope.
- There is a student on a hill, and I saw him something with my microscope.

Example 2

**Sentence: Can you help me with the can?**

In the sentence above, we observed that there are two "can" words, but they have different meanings. Here.

The first "can" word is used to form a question.

The second "can" word that is used at the end of the sentence is used to represent a container that holds some things such as food or liquid, etc.

What Conclusions we can infer from the above two Examples?

From the above two examples, we can observe that language processing is not "deterministic" that is the same language has the same interpretations, and something suitable to one person might not be suitable to another person. Therefore, Natural Language Processing (NLP) has a non-deterministic approach.

In simple words, we can use Natural Language Processing to create a new intelligent or AI system that can understand in the same way as that of humans and interpret the language in different situations.

### Difference between Rule-based NLP and Statistical NLP

Natural Language Processing is separated into two different approaches:

### Rule-based Natural Language Processing

It uses common sense reasoning for processing tasks.

**For Example,**

- The freezing temperature can lead to death, or
- Hot coffee can burn people's skin,
- Some other common-sense reasoning tasks, etc.

However, these process can take more amount of time, and it requires manual effort.

### Statistical Natural Language Processing

This type of NLP uses large amounts of data and aims to derive conclusions from it. To train NLP models, it uses machine learning algorithms. After completion of the training process on large amounts of data, the trained model will have positive outcomes with deduction.

## THE MAIN NLP APPROACHES


Source: Top 5 Semantic Technology Trends to Look for in 2017 (ontotext).

We have previously discussed a number of introductory topics in natural language processing (NLP), and I had planned at this point to move forward with covering some some useful, practical applications. It then came to my attention that I had overlooked a couple of important introductory discussions which were likely needed prior to moving ahead. The first of those topics is will be covered here, more of a general discussion on the approaches to natural language processing tasks which are common today. This also brings up the question: specifically, what types of NLP tasks are there?

The other fundamental topic which has been pointed out to me as being overlooked will be covered next week, and that will be address the common question of "how do we represent text for machine learning systems?"

But first, back to today's topic. Let's have a look at the main approaches to NLP tasks that we have at our disposal. We will then have a look at the concrete NLP tasks we can tackle with said approaches.

Approaches to NLP Tasks

While not cut and dry, there are 3 main groups of approaches to solving NLP tasks.

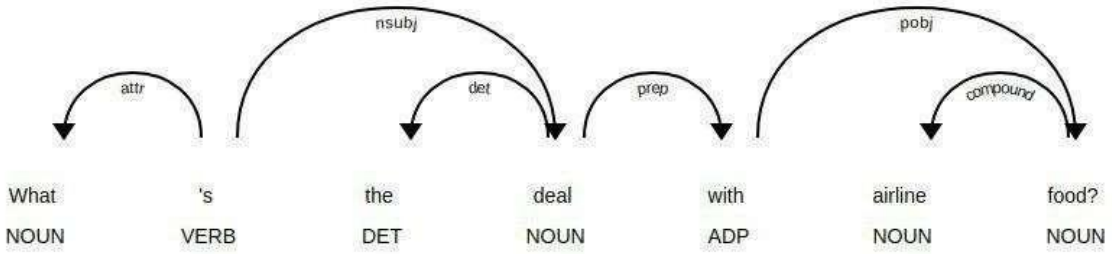



Fig 1. Dependency parse tree using spaCy.

***1. Rule-based***

Rule-based approaches are the oldest approaches to NLP. Why are they still used, you might ask? It's because they are tried and true, and have been proven to work well. Rules applied to text can offer a lot of insight: think of what you can learn about arbitrary text by finding what words are nouns, or what verbs end in -ing, or whether a pattern recognizable as Python code can be identified. Regular expressions and context free grammars are textbook examples of rule-based approaches to NLP. Rule-based approaches:

- tend to focus on pattern-matching or parsing
- can often be thought of as "fill in the blanks" methods
- are low precision, high recall, meaning they can have high performance in specific use cases, but often suffer performance degradation when generalized

***2. "Traditional" Machine Learning***

"Traditional" machine learning approaches include probabilistic modeling, likelihood maximization, and linear classifiers. Notably, these are not neural network models (see those below).

Traditional machine learning approaches are characterized by:

- training data - in this case, a corpus with markup
- feature engineering - word type, surrounding words, capitalized, plural, etc.
- training a model on parameters, followed by fitting on test data (typical of machine learning systems in general)
- inference (applying model to test data) characterized by finding most probable words, next word, best category, etc.
- "semantic slot filling"

### ***3.*** **Neural Networks**

This is similar to "traditional" machine learning, but with a few differences:

- feature engineering is generally skipped, as networks will "learn" important features (this is generally one of the claimed big benefits of using neural networks for NLP)
- instead, streams of raw parameters ("words" -- actually vector representations of words) without engineered features, are fed into neural networks
- very large training corpus

Specific neural networks of use in NLP include recurrent neural networks (RNNs) and convolutional neural networks (CNNs)



Main NLP use cases

Natural language processing or NLP is a branch of Artificial Intelligence that gives machines the ability to understand natural human speech. Using linguistics, statistics, and machine learning, computers not only derive meaning from what's said or written, they can also catch contextual nuances and a person's intent and sentiment in the same way humans do.

So, what is possible with NLP? Here are some big text processing types and how they can be applied in real life.

**Text classification**

Both in daily life and in business, we deal with massive volumes of unstructured text data: emails, legal documents, product reviews, tweets, etc. all come in a diversity of formats, which makes it hard to store and make use of. Hence, many companies fail to derive value from it.

Text classification is one of NLP's fundamental techniques that helps organize and categorize text, so it's easier to understand and use. For example, you can label assigned tasks by urgency or automatically distinguish negative comments in a sea of all your feedback.

Some common applications of text classification include the following.

Sentiment analysis. Here, text is classified based on an author's feelings, judgments, and opinion. Sentiment analysis helps brands learn what the audience or employees think of their company or product, prioritize customer service task

Spam detection. ML-based spam detection technologies can filter out spam emails from authentic ones with minimum errors. Besides simply looking for email addresses associated with spam, these systems notice slight indications of spam emails, like bad grammar and spelling, urgency, financial language, and so on.

Language detection. Often used for directing customer requests to an appropriate team, language detection highlights the languages used in emails and chats. This can be also helpful in spam and fraud detection as language is sometimes changed to conceal suspicious activity.

**Information extraction**

Another way to handle unstructured text data using NLP is information extraction (IE). IE helps to retrieve predefined information such as a person's name, a date of the event, phone number, etc., and organize it in a database.

Information extraction is a key technology in many high level tasks including:

Machine translation. Translation tools such as Google Translate rely on NLP not to just replace words in one language with words of another, but to provide contextual meaning and capture the tone and intent of the original text.

Intelligent document processing. Intelligent Document Processing is a technology that automatically extracts data from diverse documents and transforms it into the needed format. It employs NLP and computer vision to detect valuable information from the document, classify it, and extract it into a standard output format.

Question answering. Virtual assistants like Siri and Alexa and MLbased chatbots pull answers from unstructured sources for questions posed in natural language. Such dialog systems are the hardest to pull off and are considered an unsolved problem in NLP. Which of course means that there's an abundance of research in this area.

## Python Dictionary – How To Create Dictionaries In

Dictionaries in Python are a list of items that are unordered and can be changed by use of built in methods. Dictionaries are used to create a map of unique keys to values.

About Dictionaries in Python

To create a Dictionary, use {} curly brackets to construct the dictionary and [] square brackets to index it.

Separate the key and value with colons : and with commas , between each pair.

Keys must be quoted, for instance: "title" :

How to use Dictionaries in Python

As with lists we can print out the dictionary by printing the reference to it.

A dictionary maps a set of objects (keys) to another set of objects (values) so you can create an unordered list of objects.

Dictionaries are mutable, which means they can be changed.The values that the keys point to can be any Python value.

Dictionaries are unordered, so the order that the keys are added doesn't necessarily reflect what order they may be reported back. Because of this, you can refer to a value by its key name.

Create a new dictionary

# In order to construct a dictionary you can start with an empty one.

```
>>> mydict={}
```

# This will create a dictionary, which has an initially six key-value pairs, where iphone* is the key and years the values

```
released = {
            "iphone" : 2007,
            "iphone 3G" : 2008,
            "iphone 3GS" : 2009,
            "iphone 4" : 2010,
            "iphone 4S" : 2011,
            "iphone 5" : 2012
      }
print released


>>Output
{'iphone 3G': 2008, 'iphone 4S': 2011, 'iphone 3GS':
2009, '
      iphone': 2007, 'iphone 5': 2012, 'iphone 4': 2010}
```

## Add a value to the dictionary

You can add a value to the dictionary in the example below. In addition, we'll go ahead and change the value to show how dictionaries differ from a Python Set.

#the syntax is: mydict[key] = "value"

```
released["iphone 5S"] = 2013
print released
>>Output
{'iphone 5S': 2013, 'iphone 3G': 2008, 'iphone 4S': 2011,
'iphone 3GS': 2009,
'iphone': 2007, 'iphone 5': 2012, 'iphone 4': 2010}
```

## Remove a key and it's value

You can remove an element with the del operator

```
del released["iphone"]
print released
>>output
{'iphone 3G': 2008, 'iphone 4S': 2011, 'iphone 3GS': 2009, 'iphone 5': 2012,
'iphone 4': 2010}
```

## Check the length

The len() function gives the number of pairs in the dictionary. In other words, how many items are in the dictionary.

```
print len(released)
```

## Test the dictionary

Check if a key exists in a given dictionary by using the in operator like this:

```
>>> my_dict = {'a' : 'one', 'b' : 'two'}
>>> 'a' in my_dict
True
>>> 'b' in my_dict
True
>>> 'c' in my_dict
False
```

or like this in a for loop

```
for item in released:
    if "iphone 5" in released:
        print "Key found"
        break
    else:
        print "No keys found"
>>output
Key found
```

## Get a value of a specified key

```
print released.get("iphone 3G", "none")
```

## Print all keys with a for loop

```
print "-" * 10
print "iphone releases so far: "
print "-" * 10
for release in released:
    print release

>>output

iphone releases so far:
----------
iphone 3G
iphone 4S
iphone 3GS
iphone
iphone 5
iphone 4
```

Best Natural Language Processing Tools

Basically, you can start using NLP tools through SaaS (software as a service) tools or open-source libraries.

SaaS tools are ready-to-use and powerful cloud-based solutions that can be implemented with low or no code. SaaS platforms often offer pre-trained NLP models that can be used code-free, and APIs that are geared more towards those who want a more flexible, low-code, option, e.g. professional developers, or those learning to code, who want to simplify their work. If you are looking to implement NLP in a way that's cost-effective and quick, SaaS tools are the way to go!

Open-source libraries, on the other hand, are free, flexible, and allow you to fully customize your NLP tools. They are aimed at developers, however, so they're fairly complex to grasp and you will need experience in machine learning to build opensource NLP tools. Luckily, though, most of them are community-driven frameworks, so you can count on plenty of support.

To build your own NLP models with open-source libraries, you'll need time to build infrastructures from scratch, and you'll need money to invest in devs if you don't already have an in-house team of experts.

Now that you have an idea of what's available, tune into our list of top SaaS tools and NLP libraries.

**The Top 10 NLP Tools**

1. MonkeyLearn | NLP made simple
2. Aylien | Leveraging news content with NLP
3. IBM Watson | A pioneer AI platform for businesses
4. Google Cloud NLP API | Google technology applied to NLP
5. Amazon Comprehend | An AWS service to get insights from text
6. NLTK | The most popular Python library
7. Stanford Core NLP | Stanford's fast and robust toolkit
8. TextBlob | An intuitive interface for NLTK
9. SpaCy | Super-fast library for advanced NLP tasks
10. GenSim | State-of-the-art topic modeling

**1. MonkeyLearn**

MonkeyLearn is a user-friendly, NLP-powered platform that helps you gain valuable insights from your text data.

To get started, you can try one of the pre-trained models, to perform text analysis tasks such as sentiment analysis, topic classification, or keyword extraction. For more accurate insights, you can build a customized machine learning model tailored to your business.

Once you've trained your models to deliver accurate insights, you can connect your text analysis models to your favorite apps (like Google Sheets, Zendesk, Excel or Zapier) using our integrations (no coding skills needed!), or through MonkeyLearn's APIs, available in all major programming languages.

**2. Aylien**

Aylien is a SaaS API that uses deep learning and NLP to analyze large volumes of text-based data, such as academic publications, real-time content from news outlets and social media data.

You can use it for NLP tasks like text summarization, article extraction, entity extraction, and sentiment analysis, among others.

**3. IBM Watson**

IBM Watson is a suite of AI services stored in the IBM Cloud. One of its key features is Natural Language Understanding, which allows you to identify and extract keywords, categories, emotions, entities, and more.

It's versatile, in that it can be tailored to different industries, from healthcare to finance, and has a trove of documents to help you get started.

**4. Google Cloud**

The Google Cloud Natural Language API provides several pre-trained models for sentiment analysis, content classification, and entity extraction, among others. Also, it offers AutoML Natural Language, which allows you to build customized machine learning models.

As part of the Google Cloud infrastructure, it uses Google question-answering and language understanding technology.

**5. Amazon Comprehend**

Amazon Comprehend is an NLP service, integrated with the Amazon Web Services infrastructure. You can use this API for NLP tasks such as sentiment analysis, topic modeling, entity recognition, and more.

For those that work in healthcare, there's a specialized variant: the Amazon Comprehend Medical, which allows you to perform advanced analysis of medical data using Machine Learning.

**6. NLTK**

The Natural Language Toolkit (NLTK) with Python is one of the leading tools in NLP model building. Focused on research and education in the NLP field, NLTK is bolstered by an active community, as well as a range of tutorials for language processing, sample datasets, and resources that include a comprehensive Language Processing and Python handbook.

Although it takes a while to master this library, it's considered an amazing playground to get hands-on NLP experience. With a modular structure, NLTK provides plenty of components for NLP tasks, like tokenization, tagging, stemming, parsing, and classification, among others.

### 7. Stanford Core NLP

Stanford Core NLP is a popular library built and maintained by the NLP community at Stanford University. It's written in Java – so you'll need to install JDK on your computer – but it has APIs in most programming languages.

The Core NLP toolkit allows you to perform a variety of NLP tasks, such as part-ofspeech tagging, tokenization, or named entity recognition. Some of its main advantages include scalability and optimization for speed, making it a good choice for complex tasks.

### 8. TextBlob

TextBlob is a Python library that works as an extension of NLTK, allowing you to perform the same NLP tasks in a much more intuitive and user-friendly interface. Its learning curve is more simple than with other open-source libraries, so it's an excellent choice for beginners, who want to tackle NLP tasks like sentiment analysis, text classification, part-of-speech tagging, and more.

### 9. SpaCy

One of the newest open-source Natural Language Processing with Python libraries on our list is SpaCy. It's lightning-fast, easy to use, well-documented, and designed to support large volumes of data, not to mention, boasts a series of pretrained NLP models that make your job even easier. Unlike NLTK or CoreNLP, which display a number of algorithms for each task, SpaCy keeps its menu short and serves up the best available option for each task at hand.

This library is a great option if you want to prepare text for deep learning, and excels at extraction tasks. For the moment, it's only available in English.

## NLP LIBRARIES

### 1. Natural Language Toolkit (NLTK)

NLTK is one of the leading platforms for building Python programs that can work with human language data. It presents a practical introduction to programming for language processing.
NLTK comes with a host of text processing libraries for sentence detection, tokenization, lemmatization, stemming, parsing, chunking, and POS tagging.
NLTK provides easy-to-use interfaces to over 50 corpora and lexical resources. The tool has the essential functionalities required for almost all kinds of natural language processing tasks with Python.

**2. Gensim**

Gensim is a Python library designed specifically for "topic modeling, document indexing, and similarity retrieval with large corpora." All algorithms in Gensim are memory-independent, w.r.t., the corpus size, and hence, it can process input larger than RAM. With intuitive interfaces, Gensim allows for efficient multicore implementations of popular algorithms, including online Latent Semantic Analysis (LSA/LSI/SVD), Latent Dirichlet Allocation (LDA), Random Projections (RP), Hierarchical Dirichlet Process (HDP) or word2vec deep learning. Gensim features extensive documentation and Jupyter Notebook tutorials. It largely depends on NumPy and SciPy for scientific computing. Thus, you must install these two Python packages before installing Gensim.

**3. CoreNLP**

Stanford CoreNLP comprises of an assortment of human language technology tools. It aims to make the application of linguistic analysis tools to a piece of text easy and efficient. With CoreNLP, you can extract all kinds of text properties (like named-entity recognition, part-of-speech tagging, etc.) in only a few lines of code.

Since CoreNLP is written in Java, it demands that Java be installed on your device. However, it does offer programming interfaces for many popular programming languages, including Python. The tool incorporates numerous Stanford's NLP tools like the parser, sentiment analysis, bootstrapped pattern learning, part-of-speech (POS) tagger, named entity recognizer (NER), and coreference resolution system, to name a few. Furthermore, CoreNLP supports four languages apart from English – Arabic, Chinese, German, French, and Spanish.

**4. spaCy**

spaCy is an open-source NLP library in Python. It is designed explicitly for production usage – it lets you develop applications that process and understand huge volumes of text. spaCy can preprocess text for Deep Learning. It can be be used to build natural language understanding systems or information extraction systems. spaCy is equipped with pre-trained statistical models and word vectors. It can support tokenization for over 49 languages. spaCy boasts of state-of-the-art speed, parsing, named entity recognition, convolutional neural network models for tagging, and deep learning integration.

**5. TextBlob**

TextBlob is a Python (2 & 3) library designed for processing textual data. It focuses on providing access to common text-processing operations through familiar interfaces. TextBlob objects can be treated as Python strings that are trained in Natural Language Processing.

TextBlob offers a neat API for performing common NLP tasks like partof-speech tagging, noun phrase extraction, sentiment analysis, classification, language translation, word inflection, parsing, n-grams, and WordNet integration.

**6. Pattern**

Pattern is a text processing, web mining, natural language processing, machine learning, and network analysis tool for Python. It comes with a host of tools for data mining (Google, Twitter, Wikipedia API, a web crawler, and an HTML DOM parser), NLP (part-of-speech taggers, n-gram search, sentiment analysis, WordNet), ML (vector space model, clustering, SVM), and network analysis by graph centrality and visualization.

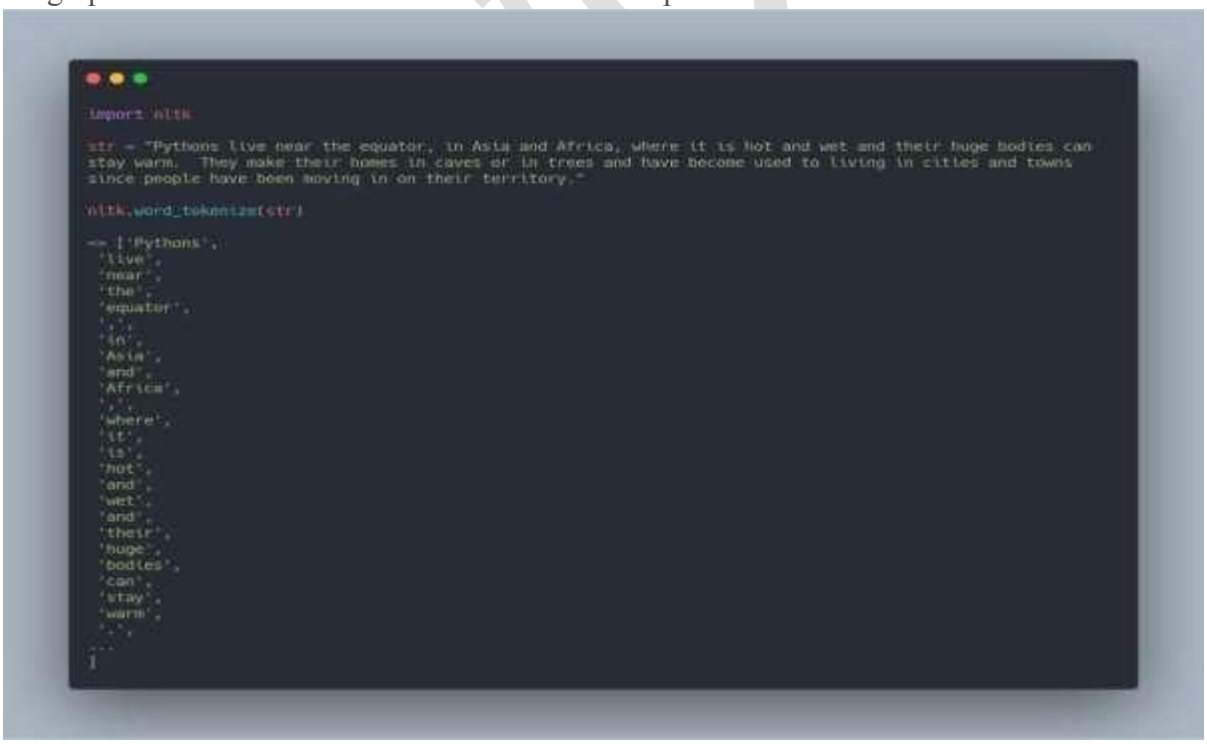
**Natural Language Processing With NLTK And Spacy**

```
# NLTK

import nltk

 # spaCy

import spacy

nlp = spacy.load(“en”)
```

To get started, create a new file like nlptest.py and import our libraries:

**Tokenization**

In the natural language processing domain, the term tokenization means to split a sentence or paragraph into its constituent words.Here’s how it’s performed withnltk




And here’s how to perform tokenization with spaCy:

**Parts Of Speech (POS) Tagging**

With POS tagging, each word in a phrase is tagged with the appropriate part of speech. Since words change their POS tag with context, there's been a lot of research in this field.

Here's what POS tagging looks like in NLTK:

And here's how POS tagging works with spaCy:


You can see how useful spaCy's object oriented approach is at this stage. Instead of an array of objects, spaCy returns an object that carries information about POS, tags, and more.

## Text Preprocessing in Python | Set – 1

Prerequisites: Introduction to NLP
Whenever we have textual data, we need to apply several pre-processing steps to the data to transform words into numerical features that work with machine learning algorithms. The pre-processing steps for a problem depend mainly on the domain and the problem itself, hence, we don't need to apply all steps to every problem.
In this article, we are going to see text preprocessing in Python. We will be using the NLTK (Natural Language Toolkit) library here.

**Python3**

```
# import the necessary libraries
import nltk import string import re
```

### Text Lowercase:

We lowercase the text to reduce the size of the vocabulary of our text data.

**Python3**

```
def text_lowercase(text):     return
text.lower()


input_str = "Hey, did you know that the summer break is coming?
Amazing right !! It's only 5 more days !!" text_lowercase(input_str)
```

**Example:**

***Input:*** *"Hey, did you know that the summer break is coming? Amazing right!! It's only 5 more days!!"*
***Output:*** *"hey, did you know that the summer break is coming? amazing right!!
it's only 5 more days!!"*

Remove numbers:

We can either remove numbers or convert the numbers into their textual representations. We can use regular expressions to remove the numbers.

**Python3**

```
#          Remove          numbers          def
remove_numbers(text):               result   =
re.sub(r'\d+', '', text)  return result


input_str = "There are 3 balls in this bag, and 12 in the other one."
remove_numbers(input_str)
```

**Example:**

***Input:*** *"There are 3 balls in this bag, and 12 in the other one."*
***Output:*** *'There are balls in this bag, and in the other one.'*

We can also convert the numbers into words. This can be done by using the inflect library.

**Python3**

```
# import the inflect library import inflect p = inflect.engine()


# convert number into words def
convert_number(text):
    # split string into list of words
temp_str = text.split() # initialise empty
list new_string = []
    for word in temp_str:
        # if word is a digit, convert the digit
        # to numbers and append into the new_string list if
word.isdigit():
            temp = p.number_to_words(word)
new_string.append(temp)
        # append the word as it is
                                        else
            :new_string.append(word)
    # join the words of new_string to form a string
temp_str = ' '.join(new_string)     return temp_str


input_str = 'There are 3 balls in this bag, and 12 in the other one.'
convert_number(input_str)
```

**Example:**

***Input:*** *"There are 3 balls in this bag, and 12 in the other one."*
***Output:*** *"There are three balls in this bag, and twelve in the other one."*

## Remove punctuation:

We remove punctuations so that we don't have different forms of the same word. If we don't remove the punctuation, then been. been, been! will be treated separately.

**Python3**

```
# remove punctuation def remove_punctuation(text):       translator =
str.maketrans('', '', string.punctuation)     return text.translate(translator)


input_str = "Hey, did you know that the summer break is coming?
Amazing right !! It's only 5 more days !!"
remove_punctuation(input_str)
```

**Example:**

***Input:*** *"Hey, did you know that the summer break is coming? Amazing right!! It's only 5 more days!!"*
***Output:*** *"Hey did you know that the summer break is coming Amazing right Its only 5 more days"*

## Remove whitespaces:

We can use the join and split function to remove all the white spaces in a string.

**Python3**

```
# remove whitespace from text def

remove_whitespace(text):     return "

".join(text.split())


input_str = " we don't need the given questions"
remove_whitespace(input_str)
```

**Example:**

**Input:** "  we don't need   the given questions"
**Output:** "we don't need the given questions"

## Remove default stopwords:

Stopwords are words that do not contribute to the meaning of a sentence. Hence, they can safely be removed without causing any change in the meaning of the sentence. The NLTK library has a set of stopwords and we can use these to remove stopwords from our text and return a list of word tokens.

```
LIST OF ENGLISH STOPWORDS IN NLTK:

their, few, wasn't, has, m, or, did, isn, very, themselves, you've, you'd, do,
between, other, t, shan, yourself, does, ours, i, it, should, what, himself, so
me, itself, there, weren, most, her, mustn, hers, doesn, won, doesn't, hasn, s,
y, wouldn't, didn't, him, couldn, after, a, will, ain, than, for, being, which,
during, ll, my, isn't, its, any, hadn't, his, then, don, of, shouldn't, out, ou
r, have, such, o, nor, too, re, should've, needn't, same, she's, but, weren't,
all, against, down, don't, can, you, under, where, wouldn, only, been, aren't,
haven, that, doing, if, up, d, needn, ma, yours, shan't, wasn, because, about,
those, he, are, was, at, hasn't, over, until, had, with, you're, below, have
n't, mightn, here, own, off, both, whom, while, as, ourselves, they, further, m
ightn't, these, from, to, them, she, who, were, more, am, why, your, aren, had
n, in, won't, yourselves, no, me, didn, an, so, before, is, on, now, each, how,
be, theirs, shouldn, mustn't, above, herself, just, you'll, the, through, agai
n, once, having, by, when, myself, we, it's, this, that'll, couldn't, ve, and,
into, not,
```

**Python3**

```
from nltk.corpus import stopwords from
nltk.tokenize import word_tokenize
# remove stopwords function def
remove_stopwords(text): stop_words =
set(stopwords.words("english")) word_tokens =
word_tokenize(text)
  filtered_text = [word for word in word_tokens if word not in
stop_words]  return filtered_text


example_text = "This is a sample sentence and we are going to
remove the stopwords from this." remove_stopwords(9_text)
```

**Example:**

***Input:*** *"This is a sample sentence and we are going to remove the stopwords from this"*
***Output:*** *['This', 'sample', 'sentence', 'going', 'remove', 'stopwords']*

## NORMALIZATION

**Text normalization** is the process of transforming text into a single canonical form that it might not have had before. Normalizing text before storing or processing it allows for separation of concerns, since input is guaranteed to be consistent before operations are performed on it. Text normalization requires being aware of what type of text is to be normalized and how it is to be processed afterwards; there is no all-purpose normalization procedure.[

## Stemming:

Stemming is the process of getting the root form of a word. Stem or root is the part to which inflectional affixes (-ed, -ize, -de, -s, etc.) are added. The stem of a word is created by removing the prefix or suffix of a word. So, stemming a word may not result in actual words.
**Example:**

books ---> book looked -

--> look denied ---> deni

flies ---> fli

| | words | stemmed words |
|---|---|---|
| 0 | connect | connect |
| 1 | connected | connect |
| 2 | connection | connect |
| 3 | connections | connect |
| 4 | connects | connect |

| | words | stemmed words |
|---|---|---|
| 0 | friend | friend |
| 1 | friends | friend |
| 2 | friended | friend |
| 3 | friendly | friendli |

If the text is not in tokens, then we need to convert it into tokens. After we have converted strings of text into tokens, we can convert the word tokens into their root form. There are mainly three algorithms for stemming. These are the Porter Stemmer, the Snowball Stemmer and the Lancaster Stemmer. Porter Stemmer is the most common among them.

**Python3**

```
from nltk.stem.porter import PorterStemmer
from nltk.tokenize import word_tokenize
stemmer = PorterStemmer()
# stem words in the list of tokenized words def stem_words(text):
word_tokens = word_tokenize(text) stems =
[stemmer.stem(word) for word in word_tokens] return stems


text = 'data science uses scientific methods algorithms and many
types of processes' stem_words(text)
```

**Example:**

***Input:*** *'data science uses scientific methods algorithms and many types of processes'*
***Output:*** *['data', 'scienc', 'use', 'scientif', 'method', 'algorithm', 'and', 'mani', 'type', 'of', 'process']*

## Lemmatization:

Like stemming, lemmatization also converts a word to its root form. The only difference is that lemmatization ensures that the root word belongs to the language. We will get valid words if we use lemmatization. In NLTK, we use the WordNetLemmatizer to get the lemmas of words. We also need to provide a context for the lemmatization. So, we add the part-of-speech as a parameter.

**Python3**

```
from nltk.stem import WordNetLemmatizer from
nltk.tokenize import word_tokenize lemmatizer =
WordNetLemmatizer()
# lemmatize string def lemmatize_word(text):
word_tokens = word_tokenize(text) #
provide context i.e. part-of-speech
  lemmas = [lemmatizer.lemmatize(word, pos ='v') for word in
word_tokens]   return lemmas


text = 'data science uses scientific methods algorithms and many
types of processes' lemmatize_word(text)
```

**Example:**

***Input:*** *'data science uses scientific methods algorithms and many types of processes'*
***Output:*** *['data', 'science', 'use', 'scientific', 'methods', 'algorithms', 'and', 'many', 'type', 'of', 'process']*

## Document Assembler

As discussed before, each annotator in Spark NLP accepts certain types of columns and outputs new columns in another type (we call this *AnnotatorType*). In Spark NLP, we have the following types: *Document, token, chunk, pos, word_embeddings, date, entity, sentiment, named_entity, dependency, labeled_dependency*.

To get through the process in Spark NLP, we need to get raw data transformed into *Document* type at first. ***DocumentAssembler()*** is a special transformer that does this for us; it creates the first annotation of type *Document* which may be used by annotators down the road.

***DocumentAssembler()*** comes from ***sparknlp.base*** class and has the following settable parameters. See the full list here and the source code here.

- **setInputCol()** -> the name of the column that will be converted. We can specify only one column here. It can read either a String column or an Array[String]

- **setOutputCol()** -> optional : the name of the column in Document type that is generated. We can specify only one column here. Default is 'document'

- **setIdCol()** -> optional: String type column with id information
- **setMetadataCol()** -> optional: Map type column with metadata information

- **setCleanupMode()** -> optional: Cleaning up options, possible values:

disabled: Source kept as original. This is a default.inplace:
removes new lines and tabs.inplace_full: removes new lines and tabs but also those which were converted to strings (i.e. \n)shrink: removes new lines and tabs, plus merging multiple spaces and blank lines to a single space.shrink_full: remove new lines and tabs, including stringified values, plus shrinking spaces and blank lines.

And here is the simplest form of how we use that.

**Python**

```
from sparknlp.base import *documentAssembler = DocumentAssembler() \
        .setInputCol("text") \
        .setOutputCol("document") \                    .setCleanupMode("shrink")doc_df = documentAssembler.transform(spark_df)
```

## ANNOTATIONS

Annotations were introduced in Python 3.0 originally without any specific purpose. They were simply a way to associate arbitrary expressions to function arguments and return values.

Years later, PEP 484 defined how to add type hints to your Python code, based off work that Jukka Lehtosalo had done on his Ph.D. project, Mypy. The main way to add type hints is using annotations. As type checking is becoming more and more common, this also means that annotations should mainly be reserved for type hints.

Function Annotations
For functions, you can annotate arguments and the return value. This is done as follows:

```
def func(arg: arg_type, optarg: arg_type = default) -> return_type:
    ...
```

For arguments, the syntax is argument: annotation, while the return type is annotated using -> annotation. Note that the annotation must be a valid Python expression.

When running the code, you can also inspect the annotations. They are stored in a special .__annotations__attribute on the function:

```
>>>
```

Sometimes you might be confused by how Mypy is interpreting your type hints. For those cases, there are special Mypy expressions: reveal_type() and reveal_locals(). You can add these to your code before running Mypy, and Mypy will dutifully report which types it has inferred. For example, save the following code to reveal.py:

```
>>> import math


>>> def circumference(radius: float) -> float:
...     return 2 * math.pi * radius
...
...
>>> circumference.__annotations___
```

```
# reveal.py

import math
reveal_type(math.pi)

radius = 1
circumference = 2 * math.pi * radius
reveal_locals()
```

Next, run this code through Mypy:

```
$ mypy reveal.py
```

```
reveal.py:4: error: Revealed type is 'builtins.float'
reveal.py:8: error: Revealed local types are:
reveal.py:8: error: circumference: builtins.float reveal.py:8: error: radius:
builtins.int
```

In Python tokenization basically refers to splitting up a larger body of text into smaller lines, words or even creating words for a non-English language. The various tokenization functions in-built into the nltk module itself and can be used in programs as shown below.

## SENTENCE TOKENIZATION

**Code #1: Sentence Tokenization –** Splitting sentences in the paragraph

```
from nltk.tokenize import sent_tokenize

text = "Hello everyone. Welcome to GeeksforGeeks. You are studying
NLP article" sent_tokenize(text)
```

**Output :**
['Hello everyone.',
'Welcome to GeeksforGeeks.', 'You are
studying NLP article']

**How sent_tokenize works ?**
The sent_tokenize function uses an instance of PunktSentenceTokenizer from the nltk.tokenize.punkt module, which is already been trained and thus very well knows to mark the end and beginning of sentence at what characters and punctuation.

## WORD TOKENIZATION

**Code #4: Word Tokenization –** Splitting words in a sentence.

```
from nltk.tokenize import word_tokenize
  text = "Hello everyone. Welcome to GeeksforGeeks."
word_tokenize(text)
```

**Output :**

**['Hello', 'everyone', '.', 'Welcome', 'to', 'GeeksforGeeks', '.']**

**How *word_tokenize* works?**

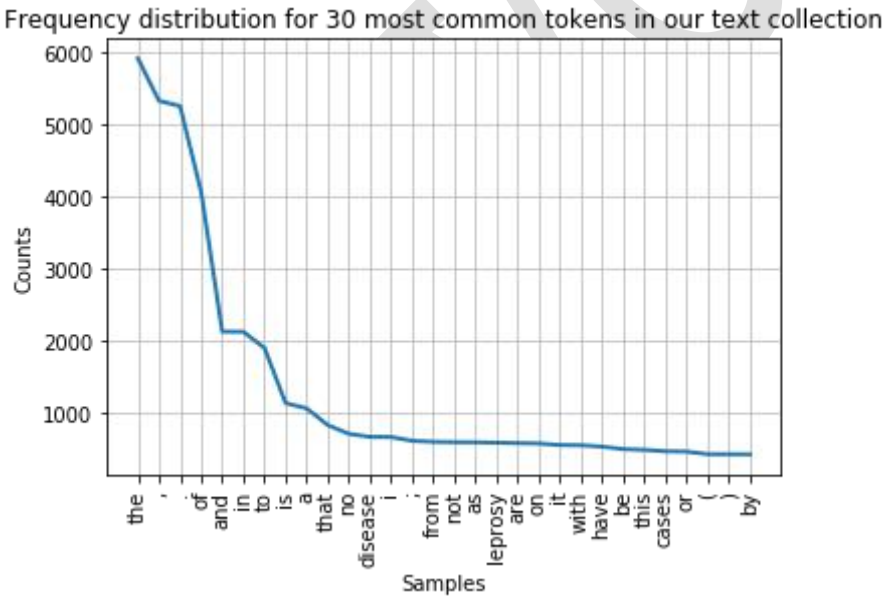
word_tokenize() function is a wrapper function that calls tokenize() on an instance of the TreebankWordTokenizer class.

## Visualising Frequency distributions of tokens in text

***Graph***

The plot() method can be called to draw the frequency distribution as a graph for the most common tokens in the text.

```
fdist.plot(30,title='Frequency distribution for 30 most common tokens in our text col
lection')
```




You can see that the distribution contains a lot of non-content words like “the”, “of”, “and” etc. (we call these stop words) and punctuation. We can remove them before drawing the graph. We need to import stopwords from the corpus package to do this. The list of stop words is combined with a list of punctuation and a list of single digits using + signs into a new list of items to be ignored.

**Note**

While it makes sense to remove stop words for this type of frequency analyis it essential to keep them in the data for other text analysis tasks. Retaining the original text is crucial, for example, when deriving part-of-speech tags of a text or for recognising names in a text. We will look at these types of text processing in day 2 of this course.

## Word cloud

We can also present the filtered tokens as a word cloud. This allows us the have an overview of the corpus using the WordCloud( ).generate_from_frequencies() method. The input to this method is a frequency dictionary of all tokens and their counts in the text. This needs to be created first by importing the Counter package in python and creating a dictionary using the filtered_text variable as input.

We generate the WordCloud using the frequency dictionary and plot the figure to size.
We can show the plot using plt.show().

```
from collections import Counter
dictionary=Counter(filtered_text)
import matplotlib.pyplot as plt from
wordcloud import WordCloud

cloud = WordCloud(max_font_size=80,colormap="hsv").generate_from_frequencies(dictiona ry)
plt.figure(figsize=(16,12))
plt.imshow(cloud, interpolation='bilinear') plt.axis('off')

plt.show()s Python3
```

STOP WORD

When working with text mining applications, we often hear of the term “stop words” or “stop word list” or even “stop list”. Stop words are basically a set of commonly used words in any language, not just English.

REMOVE WORD

Stop word removal is **one of the most commonly used preprocessing steps across different NLP applications**. The idea is simply removing the words that occur commonly across all the documents in the corpus. Typically, articles and pronouns are generally classified as stop words.

```
from nltk.corpus import stopwords from
nltk.tokenize import word_tokenize
  example_sent = """This is a sample sentence,
showing off the stop words filtration."""
  stop_words = set(stopwords.words('english'))
  word_tokens =
word_tokenize(example_sent)
  filtered_sentence = [w for w in word_tokens if not w.lower() in
stop_words]
  filtered_sentence = []
  for w in word_tokens:     if w
not in stop_words:
      filtered_sentence.append(w)
  print(word_tokens)



print(filtered_sentence)
```

Output:

['This', 'is', 'a', 'sample', 'sentence', ',', 'showing',
'off', 'the', 'stop', 'words', 'filtration', '.']
['This', 'sample', 'sentence', ',', 'showing', 'stop',

Remove Punctuation from a String with Translate

The first two arguments for string.translate method is empty strings, and the third input is a Python list of the punctuation that should be removed. This instructs the Python method to eliminate punctuation from a string. This is one of the **best ways to strip punctuation from a string**.

**Python3**

```
import string

test_str = 'Gfg, is best: for ! Geeks ;'


test_str = test_str.translate

   (str.maketrans('', '', string.punctuation)) print(test_str)
```

**Output:**

Gfg is best for  Geeks