@NANOMECH

NATURAL LANGUAGE PROCESSING

PYTHON'S NLTK

TEAM - NANO MECH

MR.R.RAJA SUBRAMANIAN

ASSOCIATE PROFESSOR
STREAM COORDINATOR - AIML
KALASALINGAM ACADEMY OF RESEARCH AND EDUCATION

- TEAM MEMBERS

- VISAL
- ESHAQ
- KRISHNA VINEETH
- LEELA SAI
- SUDHESH
- DIWAKAR

WHAT IS NLP?

- Natural language processing (NLP) refers to the branch of artificial intelligence
- This deals by allowing computers to understand the text and spoken words in much the same way human beings can.
- in simple - it is the application for the analysis and synthesis of natural language and speech.
- In NLP, the interaction, understanding, and response are made by a computer instead of a human.

WHAT IS NLP USED FOR?

- NLP is used to understand the structure and meaning of human language by analyzing different aspects like syntax, semantics, pragmatics, etc.

1. syntax - study of sentence structure.
2. semantics - study of meanings in sentences.
3. pragmatics - study of language in communication.

1. Text Classification - This involves assigning tags to texts to put them in categories. This can be useful for sentiment analysis, which helps the natural language processing algorithm determine the sentiment, or emotion behind a text.

WHAT IS NLP USED FOR?

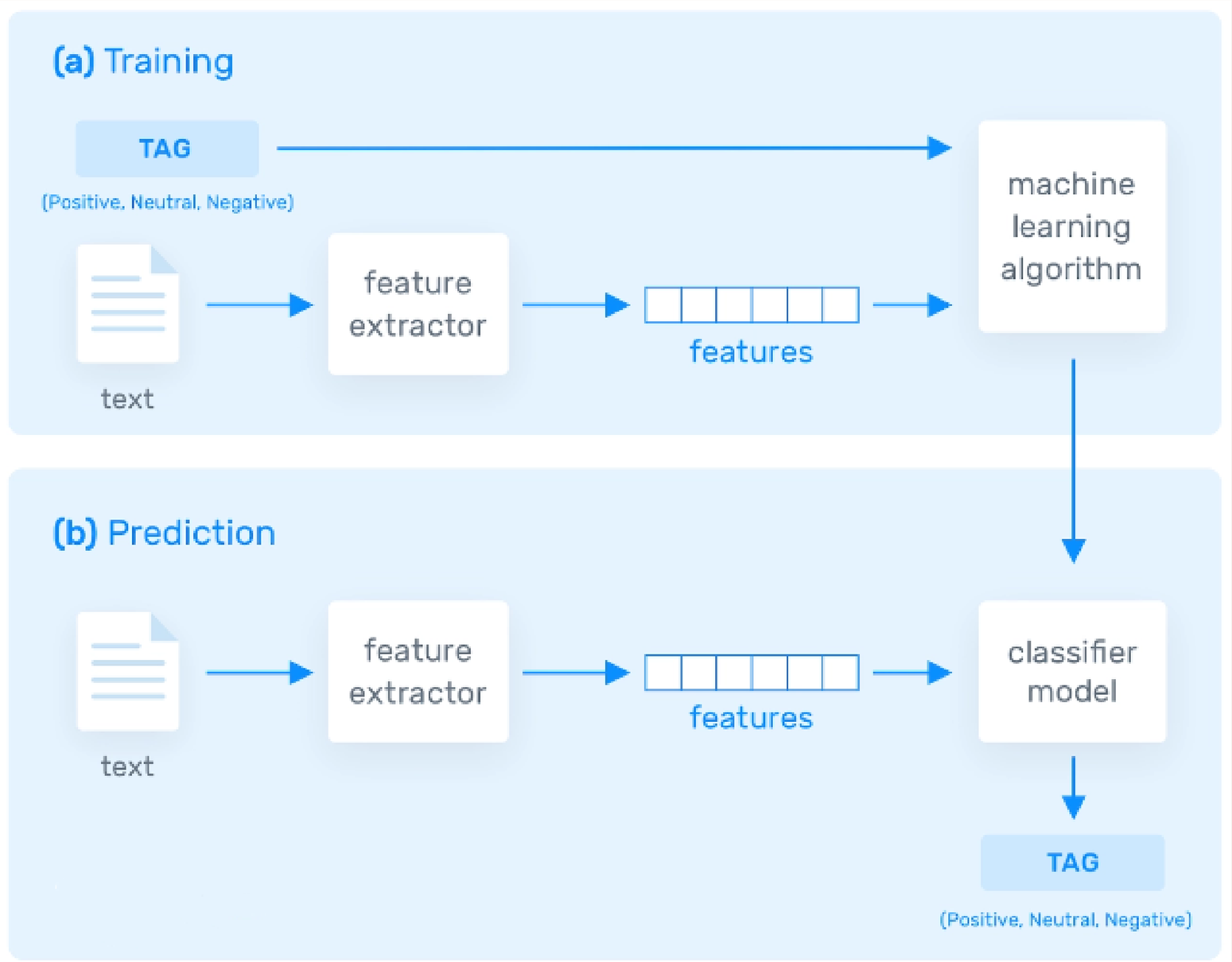
2. Text extraction - This involves automatically summarizing text and finding important pieces of data. One example of this is keyword extraction, which pulls the most important words from the text, which can be useful for search engine optimization.

3. Machine translation - This is the process by which a computer translates text from one language, such as English, to another language, such as French, without human intervention.

- For Example - Emails are automatically categorized as Promotions, Social, Primary, or Spam

- By “reading” words in subject lines and associating them with predetermined tags, machines automatically learn which category to assign emails.

HOW NLP WORKS?



There are two main phases to natural language processing:

1. **Data Preprocessing**
2. **Algorithm Development.**

1. DATA PREPROCESSING

- Data preprocessing involves preparing and "cleaning" text data for machines to be able to analyze it.
- Data preprocessing puts data in workable form and highlights features in the text that an algorithm can work with.
    - Some Steps Include:

1. **Tokenizing**
2. **Filtering Stop Words**
3. **Stemming And Lemmatizing**
4. **Tagging Parts Of Speech**
5. **Chunking And Chinking**
6. **Named Entity Recognition**

2. ALGORITHM DEVELOPMENT

- There are many different natural language processing algorithms
- But two main types are commonly used:

1. **Rules-based system -** This system uses carefully designed linguistic rules. This approach was used early on in the development of natural language processing and is still used.
2. **Machine learning-based system -** Machine learning algorithms use statistical methods. They learn to perform tasks based on training data they are fed, and adjust their methods as more data is processed. Using a combination of machine learning, deep learning, and neural networks.

COMPONENTS

There are two components of natural language processing:

1. Natural Language Understanding (NLU)
2. Natural Language Generation (NLG)

NLU	NLG
NLU is the process of reading and interpreting language.	NLG is the process of writing or generating language.
It produces non-linguistic outputs from natural language inputs.	It produces constructing natural language outputs from non-linguistic inputs.

ADVANTAGES

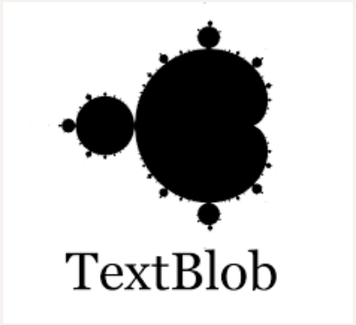
- NLP helps users to ask questions about any subject and get a direct response within seconds.
- NLP offers exact answers to the question means it does not offer unnecessary and unwanted information.
- NLP helps computers to communicate with humans in their languages.
- It is very time efficient.

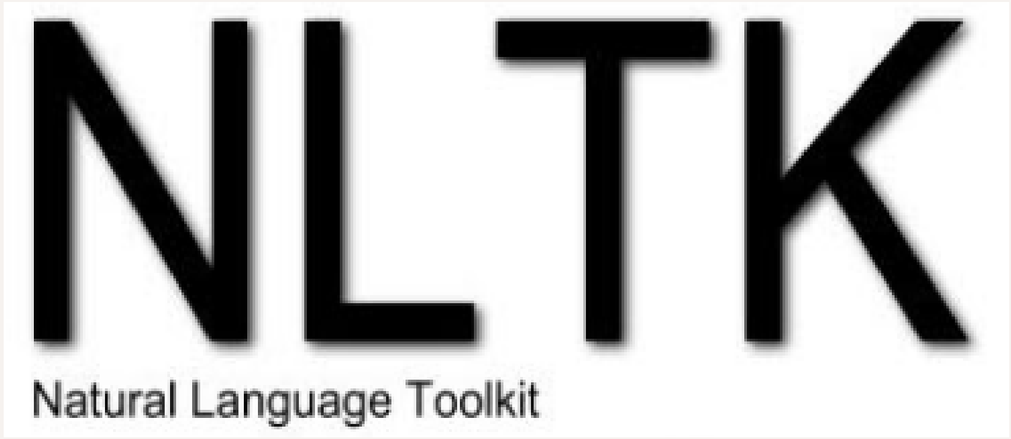
DISADVANTAGES

- The main challenge is information overload, which poses a big problem to access a specific, important piece of information from vast datasets.
- Misspellings
- NLP is unpredictable
- Development Time

NLP LIBRARIES

- **Scikit-learn**
- **Natural language Toolkit (NLTK)**
- **TextBlob**
- **Quepy**
- **SpaCy**
- **Gensim**





- The NLTK module is a massive tool kit, aimed at helping you with the entire Natural Language Processing (NLP) methodology.

- NLTK will aid you with everything from splitting sentences from paragraphs, splitting up words, recognizing the part of speech of those words, highlighting the main subjects, and then even with helping your machine to understand what the text is all about.

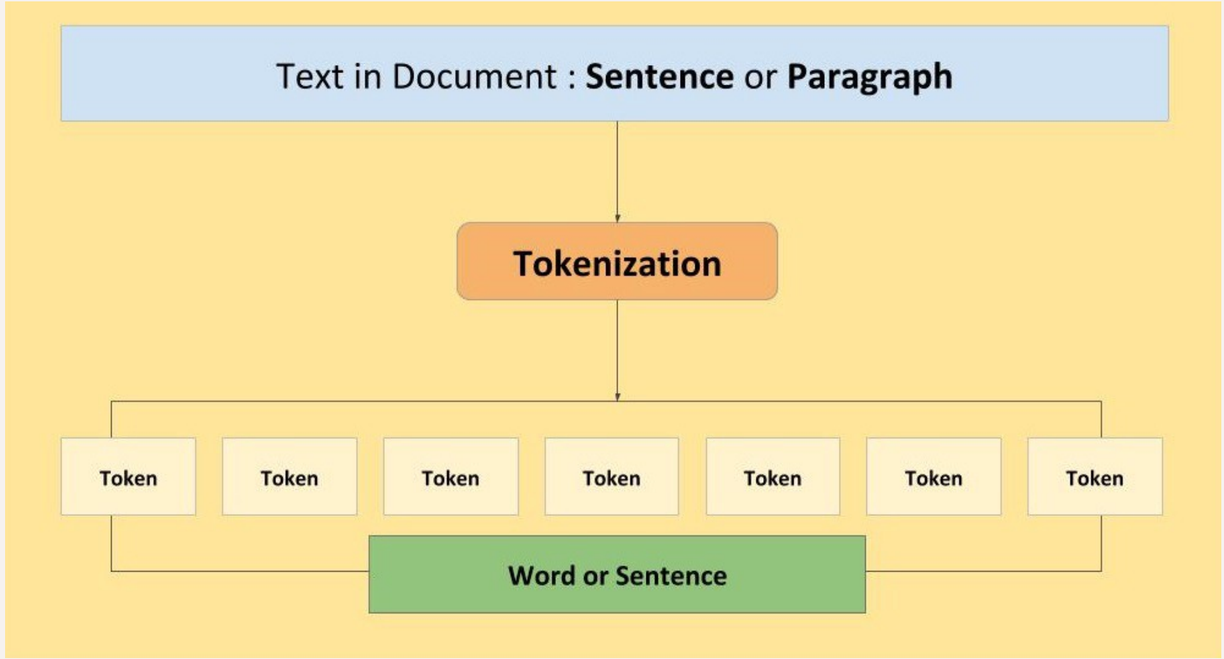
INSTALLATION

1. Firstly we need to install python idle or jupyter notebook in your system
2. "pip" is automatically installed with the idle or you can install "pip" from the official website (which is a standard package manager).
3. next check whether the latest version of pip was installed or not.
4. for installing the nltk package we need to type a command using pip in the command prompt
5. command - **"python -m pip install nltk"** or **"python -m pip install nltk=={version}"**

```
C:\Users\krish>python -m pip install nltk==3.5
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: nltk==3.5 in c:\users\krish\appdata\roaming\python\python310\site-packages (3.5)
Requirement already satisfied: click in c:\users\krish\appdata\roaming\python\python310\site-packages (from nltk==3.5) (
8.1.3)
Requirement already satisfied: joblib in c:\users\krish\appdata\roaming\python\python310\site-packages (from nltk==3.5)
(1.1.0)
Requirement already satisfied: regex in c:\users\krish\appdata\roaming\python\python310\site-packages (from nltk==3.5) (
2022.6.2)
Requirement already satisfied: tqdm in c:\users\krish\appdata\roaming\python\python310\site-packages (from nltk==3.5) (4
.64.0)
Requirement already satisfied: colorama in c:\users\krish\appdata\roaming\python\python310\site-packages (from click->nl
tk==3.5) (0.4.5)
```

TOKENIZING

- Tokenization is breaking the raw text into small chunks. Tokenization breaks the raw text into words, sentences called tokens.
- Allows you to work with smaller pieces of text.
- **Tokenizing by word:** splits a paragraph or a text into words (which are atoms of natural language)
- **Tokenizing by sentence:** splits a paragraph or a text into a group of words.



TOKENIZING IMPLEMENTATION

- first import nltk using command - **"import nltk"**.
- for tokenizing import using this Command -

  "from nltk.tokenize import sent_tokenize, word_tokenize".
- sent_tokenize for tokenizing by sentence.
- word_tokenize for tokenizing by word.

```
>>> import nltk
>>> from nltk.tokenize import sent_tokenize, word_tokenize
>>> text = """Tokenization is breaking the raw text into small chunks. Tokenization breaks the
    raw text into words, sentences called tokens."""
>>> words = word_tokenize(text)
>>> words
    ['Tokenization', 'is', 'breaking', 'the', 'raw', 'text', 'into', 'small', 'chunks', '.', '
    Tokenization', 'breaks', 'the', 'raw', 'text', 'into', 'words', ',', 'sentences', 'called'
    , 'tokens', '.']
>>> sentences = sent_tokenize(text)
>>> sentences
    ['Tokenization is breaking the raw text into small chunks.', 'Tokenization breaks the raw
    text into words, sentences called tokens.']
>>>
```

FILTERING STOP WORDS

- Stop words are words that you want to ignore, so you filter them out of your text when you’re processing it.
- Very common words like 'in', 'is', and 'an' are often used as stop words since they don’t add a lot of meaning to a text in and of themselves.

Sample text with Stop Words	Without Stop Words
GeeksforGeeks – A Computer Science Portal for Geeks	GeeksforGeeks , Computer Science, Portal ,Geeks
Can listening be exhausting?	Listening, Exhausting
I like reading, so I read	Like, Reading, read

FILTERING STOP WORDS IMPLEMENTATION

- First import nltk using command - **"import nltk"**.
- for filtering import using this Command - **"from nltk.corpus import stopwords"**.
- in case you got an error then download the stopwords data from nltk using command - **"nltk.download("stopwords")"**.

```
>>> import nltk
>>> nltk.download("stopwords")
    [nltk_data] Downloading package stopwords to
    [nltk_data]     C:\Users\krish\AppData\Roaming\nltk_data...
    [nltk_data]   Package stopwords is already up-to-date!
    True
>>> from nltk.corpus import stopwords
>>> from nltk.tokenize import word_tokenize, sent_tokenize
>>> text = """Hello I am python. I am a dynamically typed language"""
>>> words = word_tokenize(text)
>>> stop_words = set(stopwords.words("english"))
>>> filtered_list = []
>>> filtered_list = [ i for i in words if i.casefold() not in stop_words ]
>>> filtered_list
    ['Hello', 'python', '.', 'dynamically', 'typed', 'language']
>>>
```

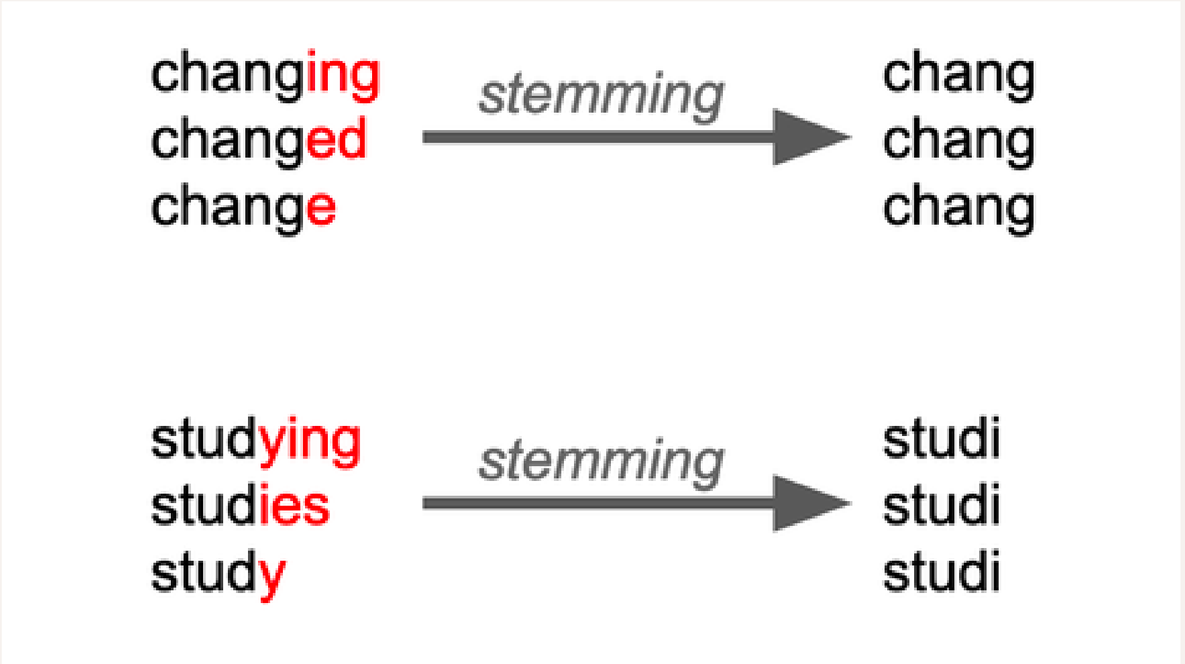
FILTERING STOP WORDS IMPLEMENTATION

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

If we want to check all the stop words provided by nltk package in the English language we can print them using command - **"set(stopwords.words("english"))".**

STEMMING

- Stemming is a text processing task in which you reduce words to their root i.e, is the core part of a word.
- For example, the words “helping” and “helper” share the root “help.”
- allows you to simplify bigger words to smaller words
- the smaller words may be not proper words
- for example: "studies" is converted to "studi".



STEMMING IMPLEMENTATION

- First import nltk using command - **"import nltk"**.
- for stemming import using this Command -

  "from nltk.stem import PorterStemmer".
- Types of stemming :

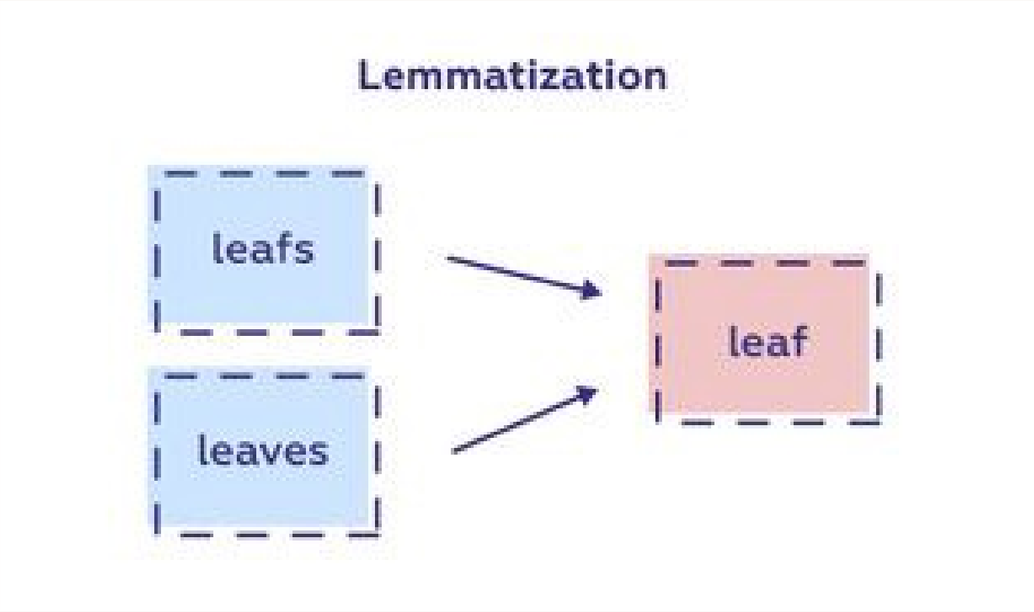
  "Porter Stemmer, Snowball Stemmer, Lancaster Stemmer, Regexp Stemmer".

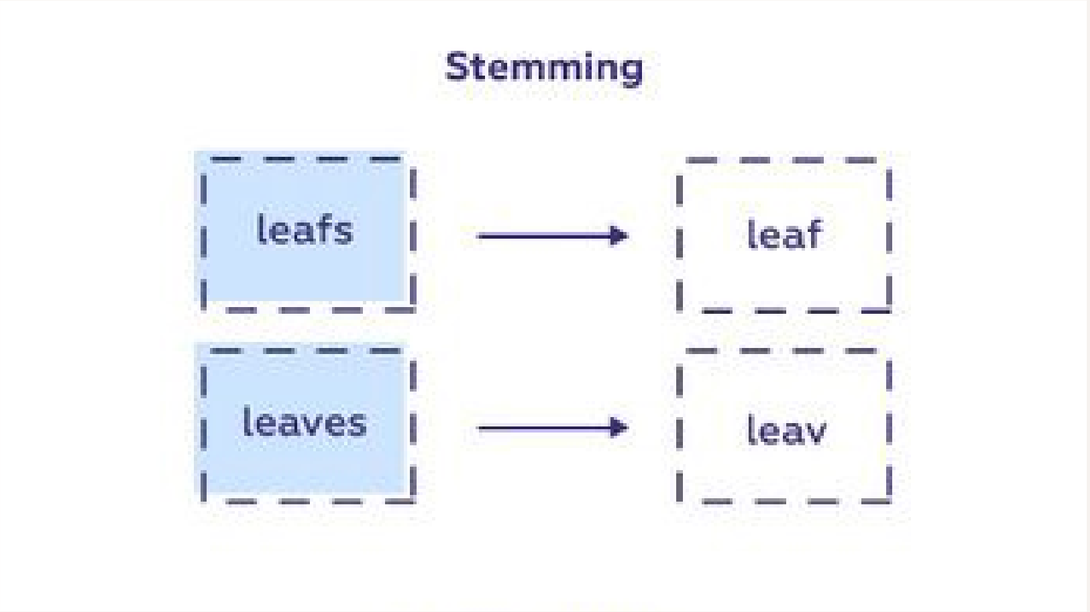
```
>>> import nltk
>>> from nltk.stem import PorterStemmer
>>> from nltk.tokenize import word_tokenize
>>> stemmer = PorterStemmer()
>>> text = """ studies and studying and study"""
>>> words = word_tokenize(text)
>>> words
['studies', 'and', 'studying', 'and', 'study']
>>> stemmed_words = [stemmer.stem(i) for i in words]
>>> stemmed_words
['studi', 'and', 'studi', 'and', 'studi']
>>> text = """discovery and discovered and discovering"""
>>> words = word_tokenize(text)
>>> words
['discovery', 'and', 'discovered', 'and', 'discovering']
>>> stemmed_words = [stemmer.stem(i) for i in words]
>>> stemmed_words
['discoveri', 'and', 'discov', 'and', 'discov']
```

LEMMATIZING

- Like stemming, lemmatizing reduces words to their core meaning,
- it will give you a complete English word that makes sense on its own
- the main difference between lemmatizing and stemming is that

"lemmatizing gives the sense full words but stemming may give the words which don't make sense".





LEMMATIZING IMPLEMENTATION

- First import nltk using command - **"import nltk"**.
- for lemmatizing import using this Command -

"from nltk.stem import WordNetLemmatizer".

```
>>> import nltk
>>> from nltk.stem import WordNetLemmatizer
>>> from nltk.tokenize import word_tokenize
>>> lemmatizer = WordNetLemmatizer()
>>> text = """leafs and leaves and leaved"""
>>> words = word_tokenize(text)
>>> words
['leafs', 'and', 'leaves', 'and', 'leaved']
>>> lemmatized_words = [lemmatizer.lemmatize(i) for i in words]
>>> lemmatized_words
['leaf', 'and', 'leaf', 'and', 'leaved']
>>>
```

LEMMATIZING IMPLEMENTATION

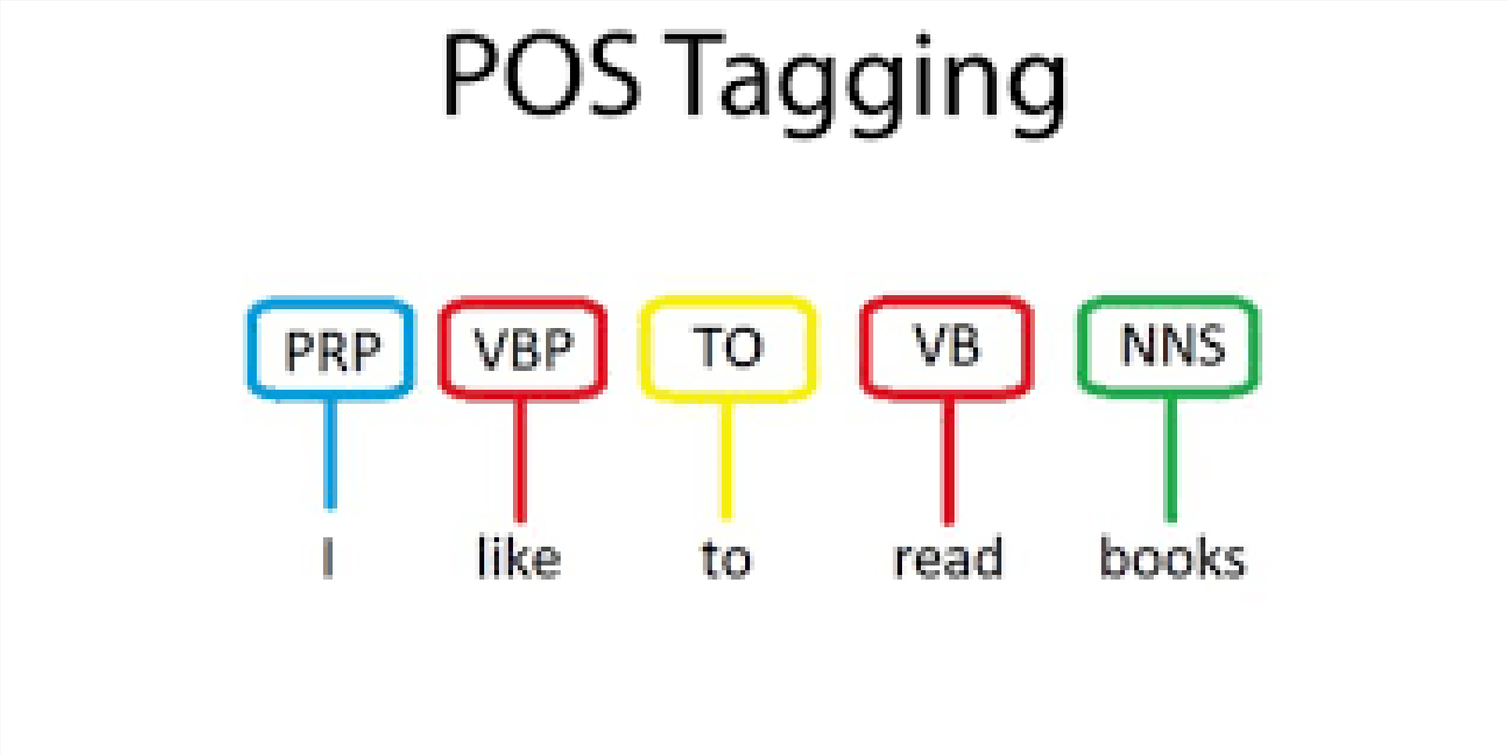
- we can use the "WordNetLemmatizer" with attributes also.
- like (parts of speech) pos = "a", that is 'a' means adjective
- for example :

    if we use the lemmatizer for the word "worst" which is a noun, if we want to get the adjective form of the word we can use pos = "a", a - adjective.

```
>>> import nltk
>>> from nltk.stem import WordNetLemmatizer
>>> lemmatizer = WordNetLemmatizer()
>>> lemmatizer.lemmatize("worst")
'worst'
>>> lemmatizer.lemmatize("worst",pos = "a")
'bad'
```

TAGGING PARTS OF SPEECH

- Part of speech is a grammatical term that deals with the roles words play when you use them together in sentences.
- Tagging parts of speech, or POS tagging, is the task of labeling the words in your text according to their part of speech.



TAGGING PARTS OF SPEECH IMPLEMENTATION

- First import nltk using command - **"import nltk"**.
- for tagging parts of speech use this Command - **"nltk.pos_tag"**.
- if you get an error the download by using command - **"nltk.download('averaged_perceptron_tagger')"**.

```
>>> import nltk
>>> nltk.download('averaged_perceptron_tagger')
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\krish\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
True
>>> from nltk.tokenize import word_tokenize
>>> text = """Hello, my name is henry, i wish to make an apple pie."""
>>> words = word_tokenize(text)
>>> words
['Hello', ',', 'my', 'name', 'is', 'henry', ',', 'i', 'wish', 'to', 'make', 'an'
, 'apple', 'pie', '.']
>>> nltk.pos_tag(words)
[('Hello', 'NNP'), (',', ','), ('my', 'PRP$'), ('name', 'NN'), ('is', 'VBZ'), ('
henry', 'JJ'), (',', ','), ('i', 'JJ'), ('wish', 'VBP'), ('to', 'TO'), ('make',
'VB'), ('an', 'DT'), ('apple', 'NN'), ('pie', 'NN'), ('.', '.')]
```

TAGGING PARTS OF SPEECH IMPLEMENTATION

- if we want to know all the pos tags present in the nltk package , we can print them using this command - **"nltk.help.upenn_tagset()"**.

```
>>> nltk.help.upenn_tagset()
$: dollar
    $ -$ --$ A$ C$ HK$ M$ NZ$ S$ U.S.$ US$
'': closing quotation mark
    ' ''
(: opening parenthesis
    ( [ {
): closing parenthesis
    ) ] }
,: comma
    ,
--: dash
    --
.: sentence terminator
    . ! ?
:: colon or ellipsis
    : ; ...
CC: conjunction, coordinating
    & 'n and both but either et for less minus neither nor or plus so
    therefore times v. versus vs. whether yet
CD: numeral, cardinal
    mid-1890 nine-thirty forty-two one-tenth ten million 0.5 one forty-
    seven 1987 twenty '79 zero two 78-degrees eighty-four IX '60s .025
    fifteen 271,124 dozen quintillion DM2,000 ...
DT: determiner
    all an another any both del each either every half la many much nary
    neither no some such that the them these this those
EX: existential there
    there
```

CHUNKING

- While tokenizing allows you to identify words and sentences, chunking allows you to identify phrases.
- One of the main goals of chunking is to group into what is known as "noun phrases."
- These are phrases of one or more words that contain a noun, maybe some descriptive words, maybe a verb, and maybe something like an adverb.
- A phrase is a word or group of words that works as a single unit to perform a grammatical function. Noun phrases are built around a noun.
- For chunking we need to know about regular expressions.
- some of them are:

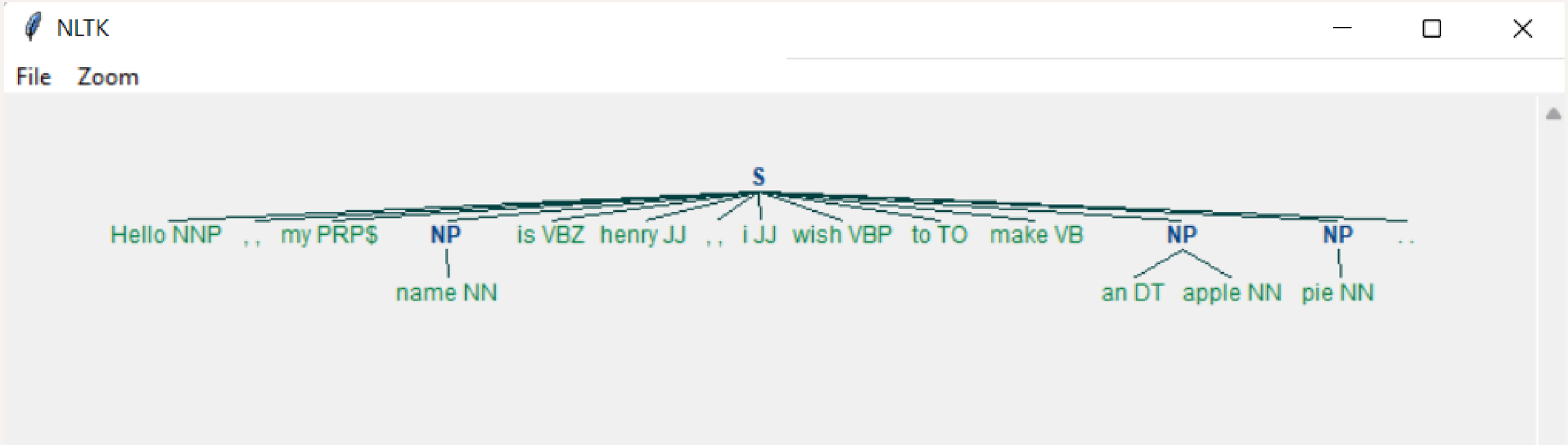
+ = match 1 or more
? = match 0 or 1 repetitions
***** = match 0 or MORE repetitions
. = Any character except a new line

CHUNKING IMPLEMENTATION

- First import nltk using command - **"import nltk"**.
- for chunking we need to create a chunk, syntax - **"NP: {pos_tags and regex}"**.
- after creating chunk we need to make a parser using this command- **"nltk.RegexpParser(var_name)"**.
- for visualization we can use **"var_name.parse(tags)"** and then **"var_name.draw()"**.

```
>>> import nltk
>>> from nltk.tokenize import word_tokenize
>>> text = """Hello, my name is henry, i wish to make an apple pie."""
>>> words = word_tokenize(text)
>>> words
['Hello', ',', 'my', 'name', 'is', 'henry', ',', 'i', 'wish', 'to', 'make', 'an'
, 'apple', 'pie', '.']
>>> text_tags = nltk.pos_tag(words)
>>> text_tags
[('Hello', 'NNP'), (',', ','), ('my', 'PRP$'), ('name', 'NN'), ('is', 'VBZ'), ('
henry', 'JJ'), (',', ','), ('i', 'JJ'), ('wish', 'VBP'), ('to', 'TO'), ('make',
'VB'), ('an', 'DT'), ('apple', 'NN'), ('pie', 'NN'), ('.', '.')]
>>> chunk = "NP: {<DT>?<JJ>*<NN>}"
>>> chunk_parser = nltk.RegexpParser(chunk)
>>> tree = chunk_parser.parse(text_tags)
>>> tree.draw()
```

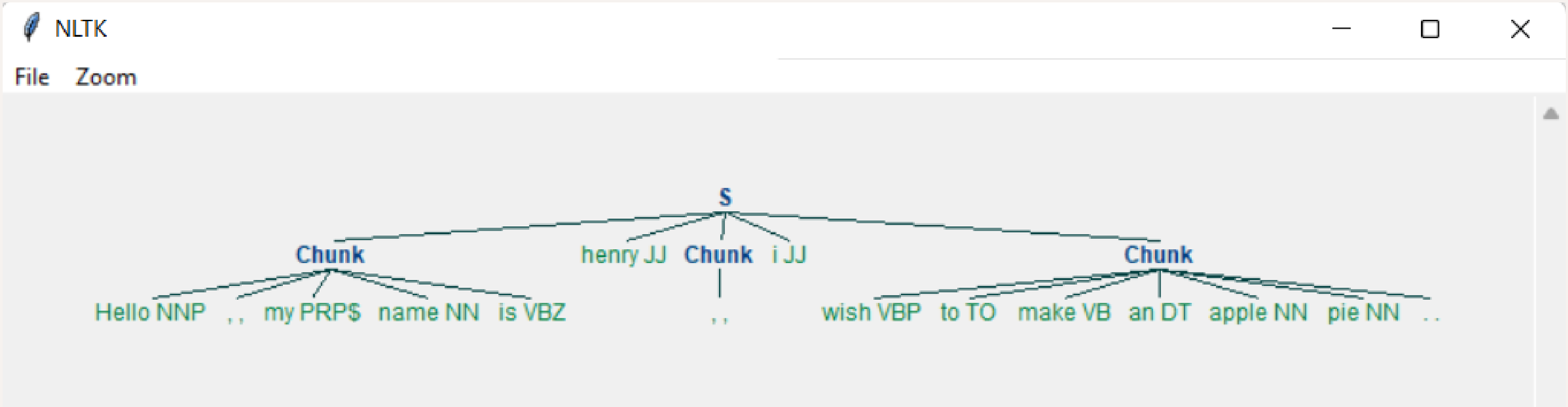
CHUNKING IMPLEMENTATION



- we can see as per the chunk created the noun phrases are created for every noun in the given text.
- that is apple is a noun before apple there is a determiner hence making a sentence.
- for the remaining nouns as per the chunk, there are no determiners and adjectives.

CHINKING

- Chinking is same as chunking
- In chunking we create the noun phrases, in chinking we create the chunks
- that is in chunking **"{pos_tags and regex}"** are included
- where as in thinking **"}pos_tags and regex{"** are excluded.



CHINKING IMPLEMENTATION

- First import nltk using command - **"import nltk"**.
- for chInking we need to create a chink, syntax - **"""**

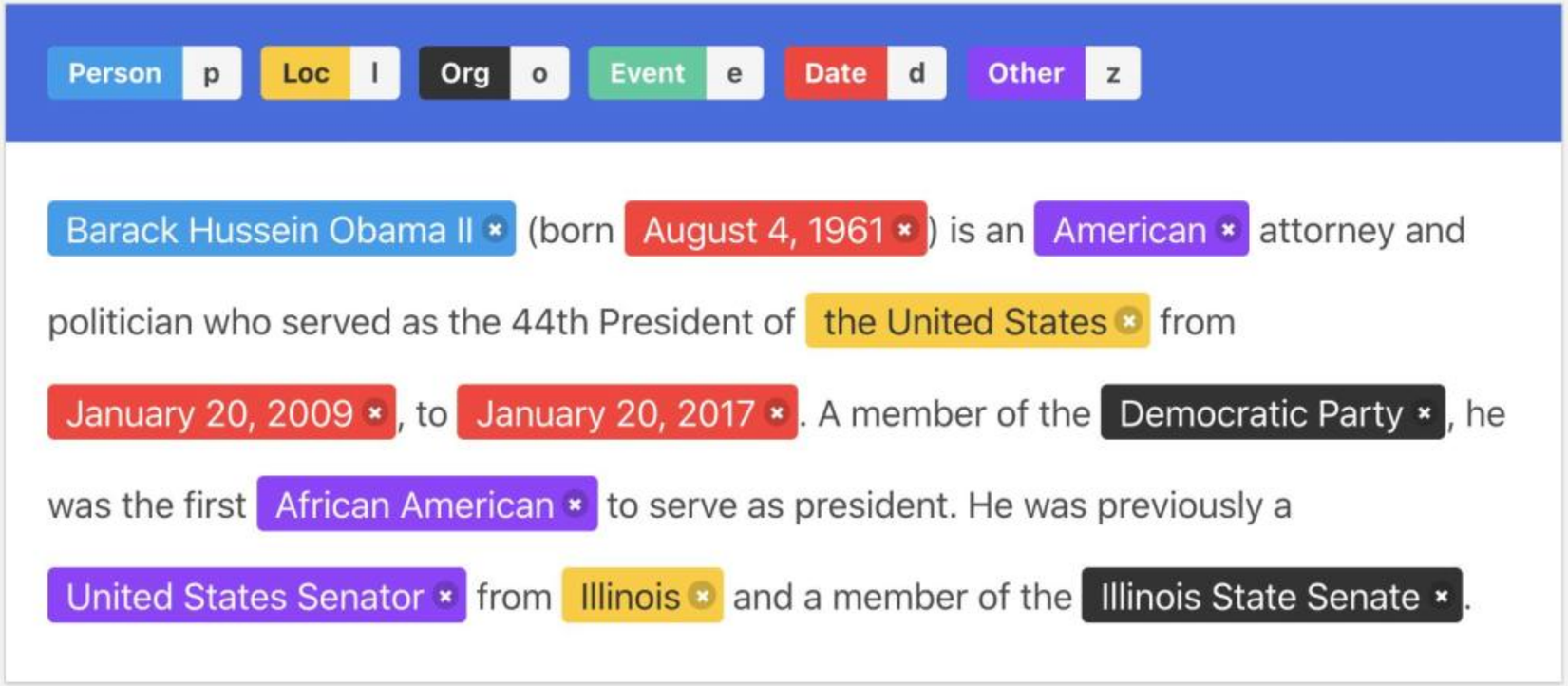
Chunk: {pos_tags and regex}

}pos_tags and regex{""".

```
>>> import nltk
>>> from nltk.tokenize import word_tokenize
>>> text = """Hello, my name is henry, i wish to make an apple pie."""
>>> words = word_tokenize(text)
>>> words
['Hello', ',', 'my', 'name', 'is', 'henry', ',', 'i', 'wish', 'to', 'make', 'an'
, 'apple', 'pie', '.']
>>> text_tags = nltk.pos_tag(words)
>>> text_tags
[('Hello', 'NNP'), (',', ','), ('my', 'PRP$'), ('name', 'NN'), ('is', 'VBZ'), ('
henry', 'JJ'), (',', ','), ('i', 'JJ'), ('wish', 'VBP'), ('to', 'TO'), ('make',
'VB'), ('an', 'DT'), ('apple', 'NN'), ('pie', 'NN'), ('.', '.')]
>>> chink = """
... Chunk: {<.*>+}
...          }<JJ>{"""
>>> chunk_parser = nltk.RegexpParser(chink)
>>> tree = chunk_parser.parse(text_tags)
>>> tree.draw()
```

Named Entity Recognition (NER)

- Named entities are noun phrases that refer to specific locations, people, organizations, and so on.
- With named entity recognition, you can find the named entities in your texts and also determine what kind of named entity they are.



- ORGANIZATION
- PERSON
- LOCATION
- DATE
- TIME
- MONEY
- PERCENT
- FACILITY
- GPE

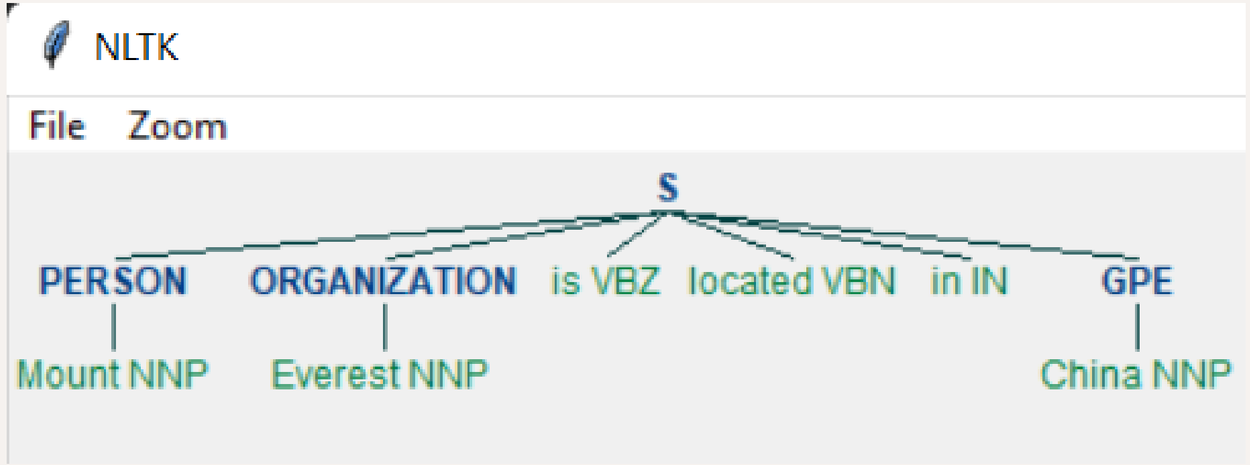
Named Entity Recognition Implementation

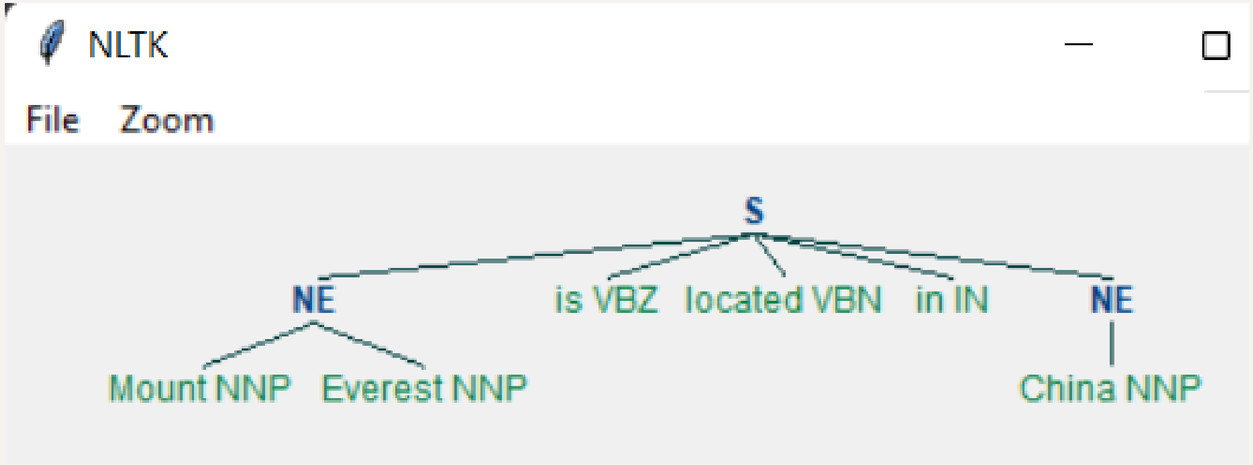
- First import nltk using command - **"import nltk"**.
- implement using this Command - **"nltk.ne_chunk(pos_tagged_words)"**.
- in case you got an error then download data from nltk using command - **"nltk.download("maxent_ne_chunker")"**.

```
>>> import nltk
>>> nltk.download("maxent_ne_chunker")
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     C:\Users\krish\AppData\Roaming\nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
True
>>> nltk.download("words")
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\krish\AppData\Roaming\nltk_data...
[nltk_data]   Package words is already up-to-date!
True
>>> from nltk.tokenize import word_tokenize
>>> text = """Mount Everest is located in China"""
>>> words = word_tokenize(text)
>>> words
['Mount', 'Everest', 'is', 'located', 'in', 'China']
>>> text_tags = nltk.pos_tag(words)
>>> text_tags
[('Mount', 'NNP'), ('Everest', 'NNP'), ('is', 'VBZ'), ('located', 'VBN'), ('in',
 'IN'), ('China', 'NNP')]
>>> tree = nltk.ne_chunk(text_tags)
```

Named Entity Recognition Implementation

- If we needed to know the exact entity then we can use it simply without attributes.
- if we needed just only which are named entities then we can use an attribute
- that is using **" binary = True ".**





CONCLUSION

Natural Language	Computer Language
Natural language has a very large vocabulary.	Computer language has a very limited vocabulary.
Natural language is easily understood by humans.	Computer language is easily understood by the machines.
Natural language is ambiguous in nature.	Computer language is unambiguous.

- SINCE THE NATURAL LANGUAGE IS AMBIGUOUS IN NATURE
- WE USE NATURAL LANGUAGE PROCESSING TECHNIQUES TO CONVERT IT INTO THE COMPUTER LANGUAGE
- AND THIS MAKES IT EASIER TO UNDER THE TEXT SINCE UNAMBIGUOUS IN NATURE.
- WITH NLP WE HAVE MANY APPLICATIONS
- SOME OF THEM ARE **"QUESTION ANSWERING, SENTIMENT ANALYSIS, SPAM DETECTION, MACHINE TRANSLATION, SPELLING CORRECTION, SPEECH RECOGNITION, CHATBOTS, INFORMATION EXTRACTION"** ETC.

PROJECT

EMAIL SPAM CLASSIFICATION USING NLTK

THANK YOU