**TKR COLLEGE OF ENGINEERING AND TECHNOLOGY**

**Medbowli, Meerpet, Saroornagar (M), Hyderabad-97**

**NATURAL LANGUAGE PROCESSING VI SEMESTER**

**(R20 REGULATION)**

# CSE (DATA SCIENCE)

CSE(DATA SCIENCE),TKRCET

# B.Tech V Semeste L/ T/P/C

**0 /0/ 2/ 1**

**NATURAL LANGUAGE PROCESSING LAB-Manual (C75PC8)**

**Course Objective:**

To introduce the students with the basics of NLP which will empower them for developing advanced NLP tools and solving practical problems in the field.

## Course Outcomes:

Upon completion, of course the student will be able to

1. Implementing experimental methodology for training and evaluating empirical NLP systems.
2. Show Sensitivity to linguistic phenomena and an ability to model them with formal grammars.
3. Design, implement, and analyze NLP algorithms
4. Design different language modeling Techniques.

## List of experiments:

1. [Word Analysis](https://nlp-iiith.vlabs.ac.in/exp/word-analysis/): experiment is to learn about morphological features of a word by analyzing it
2. [Word Generation](https://nlp-iiith.vlabs.ac.in/exp/word-generation/) : experiment is to generate word forms from root and suffix information
3. [Morphology](https://nlp-iiith.vlabs.ac.in/exp/morphology/): experiment understands the morphology of a word by the use of Add-Delete table.
4. [N-Grams](https://nlp-iiith.vlabs.ac.in/exp/n-grams/): experiment is to learn how to apply add-one smoothing on sparse bigram table.
5. [N-Grams smoothing](https://nlp-iiith.vlabs.ac.in/exp/n-grams-smoothing/): experiment is to learn how to apply add-one smoothing on sparse bigram table.
6. [POS Tagging: Hidden Markov Model](https://nlp-iiith.vlabs.ac.in/exp/markov-model/): experiment is to calculate emission and transition matrix which will be helpful for tagging Parts of Speech using Hidden Markov Model.
7. [POS Tagging: Viterbi Decoding](https://nlp-iiith.vlabs.ac.in/exp/viterbi-decoding/): experiment is to find POS tags of words in a sentence using Viterbi decoding.
8. [Building POS Tagger](https://nlp-iiith.vlabs.ac.in/exp/pos-tagger/): experiment is to know the importance of context and size of training corpus in learning Parts of Speech.
9. [Chunking](https://nlp-iiith.vlabs.ac.in/exp/chunking/): experiment is to understand the concept of chunking and get familiar with the

basic chunk tagset

1. [Building Chunker](https://nlp-iiith.vlabs.ac.in/exp/building-chunker/): experiment is to know the importance of selecting proper features for training a model and size of training corpus in learning how to do chunking.

**EXPERIMENT NO.1**

**Aim:** Experiment is to learn about morphological features of a word by analyzing it

**Theory:**

To preprocess your text simply means to bring your text into a fom1 that is predictable and analyzable for your task. A task here is a combination of approach and domain.

Machine Leaming needs data in the numeric form. We basically used encoding technique (BagOf.Vlord, Bi-gram,n-gram, TF-IDF, Word2Vec) to encode text into numeric vector. But before encoding we first need to clean the text data and this process to prepare (or clean) text databefore encoding is called text preprocessing, this is the very first step to solve the NLP problems.

**Tokenization:**

Tokenization is about splitting strings of text into smaller pieces, or "tokens". Paragraphs can be tokenized into sentences and sentences can be tokenized into words.

**Filtration:**

Similarly, if we are doing simple word counts, or trying to visualize our text with a word cloud, stopwords are some of the most frequently occurring words but don't really tell us anything. We're often better off tossing the stopwords out of the text. By checking the Filter Stopwords option in the Text Pre-processing tool, you can automatically filter these words out.

**Script Validation:**

The script must be validated properly.

# code:

import nltk

import numpy as np nltk.download("punkt")

[nltk\_data] Downloading package punkt to

[nltk\_data] C:\Users\TKRLIB\AppData\Roaming\nltk\_data... [nltk\_data] Package punkt is already up-to-date!

True

from nltk.tokenize import word\_tokenize from nltk.tokenize import sent\_tokenize

text1="this is the first sentence.this is the second sentence.is this third" print(text1)

this is the first sentence.this is the second sentence.is this third sent\_tokenize(text1)

['this is the first sentence.this is the second sentence.is this third'] len(sent\_tokenize(text1))

1

text1 = "this is the first sentence. this is the second sentence. is this third?" print(text1)

this is the first sentence. this is the second sentence. is this third? sent\_tokenize(text1)

['this is the first sentence.', 'this is the second sentence.', 'is this third?'] len(sent\_tokenize(text1))

3

word\_tokenize(text1) ['this',

'is',

'the',

'first', 'sentence', '.',

'this',

'is',

'the',

'second', 'sentence', '.',

'is',

'this',

'third',

'?']

len(word\_tokenize(text1))

16

word\_result = word\_tokenize(text1) print(word\_result)

['this', 'is', 'the', 'first', 'sentence', '.', 'this', 'is', 'the', 'second', 'sentence', '.', 'is', 'this', 'third', '?'] word\_result[1:5]

['is', 'the', 'first', 'sentence'] word\_result[:5]

['this', 'is', 'the', 'first', 'sentence']

from nltk.tokenize.punkt import PunktSentenceTokenizer pst = PunktSentenceTokenizer()

pst.tokenize(text1)

['this is the first sentence.', 'this is the second sentence.', 'is this third?'] len(pst.tokenize(text1))

3

# EXPERIMENT NO:2

**Aim**: Experiment is to generate word forms from root and suffix information.

**Theory:**

To preprocess your text simply means to bring your text into a fom1 that is predictable and analyzable for your task. A task here is a combination of approach and domain.

**Stemming**

Stemming is the process of reducing inflection in words (e.g. troubled, troubles) to their root fonn (e.g. trouble). The "root" in this case may not be a real root word, but just a canonical form of the original word.

Stemming uses a crnde heuristic process that chops off the ends of words in the hope of correctly transfom1ing words into its root fom1. So, the words "trouble", "troubled" and "troubles" might.

## Stemmed words:

actually be converted to troublinstead of trouble because the ends were just chopped off.

There are different algorithms for stemming. The most common algorithm, which is also known to be empirically effective for English, is Porters Algorithm. Here is an example of stemming in action with Porter Stemmer:

**original\_word stemmed\_words**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **0** | **connect** | **connect** |
| **1** | **connected** | **connect** |
| **2** | **connection** | **connect** |
| **3** | **connections** | **connect** |
| **4** | **connects** | **connect** |
| **Stopword Removal** |  |  |  |

Stop words are a set of commonly used words in a language. Examples of stop words in English are "a", "the", "is", "are" and etc. The intuition behind using stop words is that, by removing low infom1ation words from text, we can focus on the important words instead.

For example, in the context of a search system, if your search query is *"what is text preprocessing?",* you want the search system to focus on surfacing documents that talk about text preprocessing over documents that talk about what is. This can be done by preventing all words from your stop word list from being analyzed. Stop words are commonly applied in search systems, text classification applications, topic modeling, topic extraction and others.

In my experience, stop word removal, while effective in search and topic extraction systems, showed to be non-critical in classification systems. However, it does help reduce the number of features in consideration which helps keep your models decently sized.

Here is an example of stop word removal in action. All stop words are replaced with a dummy character, **W:**

**originalsentence• thisisa extfull of contentand we needtoclean it up**

**sentence with stopwords removed• WWW text full W content WWW W clean WW**

**Code:**

import enchant

>>> d = enchant.Dict("en\_US")

>>> tokens=[]

>>> def tokenize(st):

... if not st:return

... for i in xrange(len(st),-1,-1):

... if d.check(st[0:i]):

... tokens.append(st[0:i])

... st=st[i:]

... tokenize(st)

... break

...

>>> tokenize("societynamebank")

>>> tokens

['society', 'name', 'bank']

>>> tokens=[]

>>> tokenize("HelloSirthereissomethingwrongwiththistext")

>>> tokens

**Output:**

['Hello', 'Sir', 'there', 'is', 'something', 'wrong', 'with', 'this', 'text']

# EXPERIMENT:3

**Aim:** To Study Morphological Analysis

## Theory:

**Morphological Analysis:**

While performing the morphological analysis, each particular word is analyzed. Non word tokens such as punctuation are removed from the words. Hence the remaining words are assigned categories. For instance, Ram’s iPhone cannot convert the video from .mkv to .mp4. In Morphological analysis, word by word the sentence is analyzed.So here, Ram is a proper noun, Ram’s is assigned as possessive suffix and .mkv and .mp4 is assigned as a file extension. As shown above, the sentence is analyzed word by word. Each word is assigned a syntactic category. The file extensions are also identified present in the sentence which is behaving as an adjective in the above example. In the above example, the possessive suffix is also identified. This is a very important step as the judgment of prefixes and suffixes will depend on a syntactic category for the word. For example, swims and swims are different. One makes it plural, while the other makes it a third person singular verb. If the prefix or suffix is incorrectly interpreted then the meaning and understanding of the sentence are completely changed. The interpretation assigns a category to the word. Hence, discard the uncertainty from the word.

## Regular Expression:

Regular expressions also called regex. It is a very powerful programming tool that is used for a variety of purposes such as feature extraction from text, string replacement and other string manipulations. A regular expression is a set of characters, or a pattern, which is used to find sub strings in a given string. for ex. extracting all hashtags from a tweet, getting email id or phone numbers etc., from a large unstructured text content. In short, if there’s a pattern in any string, you can easily extract, substitute and do variety of other string manipulation operations using regular expressions. Regular expressions are a language in itself since they have their own compilers and almost all popular programming languages support working with regexes.

## Stop Word Removal:

The words which are generally filtered out before processing a natural language are called

stop words . These are actually the most common words in any language (like articles, repositions, pronouns, conjunctions, etc) and does not add much information to the text. Examples of a few stop words in English are “the”, “a”, “an”, “so”, “what”. Stop words are available in abundance in any human language. By removing these words, we remove the low level information from our text in order to give more focus to the important

information. In order words, we can say that the removal of such words does not how any negative consequences on the model we train for our task. Removal of stop words definitely reduces the dataset size and thus reduces the training time due to the fewer number of tokens involved in the training.

## Synonym:

The word synonym defines the relationship between different words that have a similar meaning. A simple way to decide whether two words are synonymous is to check for substitutability. Two Words are synonyms in a context if they can be substituted for each for each other without changing the meaning of the sentence.

## Stemming:

Stemming is the process of reducing a word to its word stem that affixes to suffixes and prefixes or to the roots of words known as a lemma. Stemming is important in atural language understanding (NLU) and natural language processing (NLP).

**Code**:

**Output:**

1. import re
2. input="The 5 biggest animals are 1.Elephant, 2Rhinoand, 3dinasaur"
3. input=input.lower()
4. print(input)
5. result=re.sub(r'\d+','',input)
6. print(result)

the 5 biggest animals are 1.elephant, 2rhinoand, 3dinasaur the biggest animals are .elephant, rhinoand, dinosaur **Synonym: (**Install **nltk** toolkit**)**

**Code**

* 1. import nltk
  2. nltk.download('wordnet')
  3. from nltk.corpus import wordnet
  4. synonyms = []
  5. for syn in wordnet.synsets(‘Machine'):
  6. for lemma in syn.lemmas():
  7. synonyms.append(lemma.name())
  8. print(synonyms)

**Output**:

['machine', 'machine', 'machine', 'machine', 'simple\_machine', 'machine', 'political\_machine', 'car', 'auto',

'automobile', 'machine', 'motorcar', 'machine', 'machine']

**Stemming:**

**Code:**

From nltk.stem import PorterStemmer

stemmer=PorterStemmer() print(stemmer.stem('eating')) print(stemmer.stem('ate'))

**Output**:

eat ate

**Conclusion:**

Thus, in the above experiment we have studied regarding morphological analysis in detail with stemming, synonym, stop word removal, regular expression and tried to implement the code and got proper output.

# EXPERIMENT:4

**Aim**: To study N gram model .Experiment is to learn how to apply add-one smoothing on sparse bigram table.

Theory: Given a sequence of N-1 words, an N - gram model predicts the most robable word that might follow this sequence. It's a probabilistic model that's trained on a corpus of text. Such a model is useful in many NLP applications including speech recognition, machine translation and predictive text input.

An N-gram model is built by counting how of ten word sequences occur in corpus text and then estimating the probabilities. Since a simple N - gram model has limitations, improvements are often made via smoothing, interpolation and backoff.

An N-gram model is one type of a Language Model (LM), which is about finding the probability distribution over word sequences. Consider two sentences: "There was heavy rain" vs. "There was heavy flood". From experience, we know that the former sentence sounds better. An N - gram model will tell us that "heavy rain" occurs much more often than "heavy flood" in the training corpus. Thus, the first sentence is more probable and will be selected by the model.

A model that simply relies on how often a word occurs without looking at previous words is called unigram . If a model considers only the previous word to predict the current word, then it's called bigram . If two previous words are considered, then it's a trigram model.

An n-gram model for the above example would calculate the following probability:

P('There was heavy rain') = P('There', 'was', 'heavy', 'rain') = P('There')P('was'|'There')P('heavy'|'There was')P('rain'|'There was heavy')

Since it's impractical to calculate these conditional probabilities, using Markov assumption , we approximate this to a bigram model:

P('There was heavy rain') ~ P('There')P('was'|'There')P('heavy'|'was')P('rain'|'heavy') In speech recognition, input may be noisy and this can lead to wrong speech

* to - text conversions. N - gram models can correct this based on their knowledge of the probabilities. Likewise, N -gram models are used in machine translation to produce more natural sentences in the target language.

The use of dictionaries to check for spelling problems is not always effective. For instance, the term "mineuts" is a dictionary-acceptable word but is wrong in the phrase "in around fifteen mineuts." N-gram models can fix these mistakes. Most N-gram models operate at the word level. Additionally, it has been used to do stemming at the character level, which separates the rootword from the suffix. N-gram statistics can be used to categorise languages or distinguish between US and UK spellings. Sz is a typical Czech sound, whereas gb and kp are typical Igbo

sounds. N-gram models are useful for a wide range of NLP applications, such as part-of-speech tagging, natural language production, word similarity, sentiment extraction, and predictive text input.

Code

import re

from nltk.util import ngrams s="MachinelearningisanimportantpartofAI""andAIisgoingtobecomeinmporantfordailyfunctionong" tokens=[tokenfortokenins.split("")]

output=list(ngrams(tokens,2)) print(output)

Output

[('Machine', 'learning'), ('learning', 'is'), ('is', 'an'), ('an', 'important'), ('important', 'part'), ('part', 'of'), ('of',

'AI'), ('AI', 'and'), ('and', 'AI'), ('AI', 'is'), ('is', 'going'), ('going', 'to'), ('to', 'become'), ('become', 'inmporant'), ('inmporant', 'for'), ('for', 'daily'), ('daily', 'functionong'), ('functionong', '')]

**Conclusion**

Thus, in the above experiment we have studied regarding N-Gram Model in detail with the help of theory and then tried to implement the code and successfully executed it.

# EXPERIMENT:5

**Aim:** experiment is to learn how to apply add-one smoothing on sparse bigram table.

## Theory:

Named Entity Recognition (NER) is a standard NLP problem which involves spotting named entities (people, places, organizations etc.) from a chunk of text, and classifying them into a predefined set of categories. Some of the practical applications of NER include:

* + Scanning news articles for the people, organizations and locations reported.
  + Providing concise features for search optimization: instead of searching the entire content, one may simply search for the major entities involved.
  + Quickly retrieving geographical locations talked about in Twitter posts.

In any text document, there are particular terms that represent specific entities that are more infom1ative and have a unique context. These entities are known as named entities, which more specifically refer to terms that represent real-world objects like people, places, organizations, and so on, which are often denoted by proper names. A naive approach could be to find these by looking at the noun phrases in text documents. Named entity recognition (NER), also known as entity chunking/extraction, is a popular technique used in infom1ation extraction to identify and segment the named entities and classify or categorize them under various predefined classes.

## Input Text

**Preprocessing Module**

***r***

--

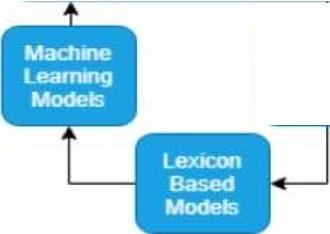


...



Tokemzation

Output.\_ Post-processing Text IVoduJe



'--- *,I*

I**R**-. **ul**"

I **'es**•

## IBased

.**M**.. -,**o**.

**d**..**e**.

**How NER works**

At the heaii of any NER model is a two step process:

Detect a named entity Categorize the entity

Beneath this lie a couple of things.

Step one involves detecting a word or string of words that fom1 an entity. Each word represents a token: "The Great Lakes" is a string of three tokens that represents one entity. Inside-outside• beginning tagging is a common way of indicating where entities begin and end. We'll explore this further in a future blog post.

The second step requires the creation of entity categories.

## How is NER used?

NER is suited to any situation in which a high-level overview of a large quantity of text is helpful. With NER, you can, at a glance, understand the subject or theme of a body of text and quickly group texts based on their relevancy or similarity.

Some notable NER use cases include:

Speed up the hiring process by summarizing applicants' CVs; improve internal workflows by categorizing employee complaints and questions

## Customer support

Improve response times by categorizing user requests, complaints and questions and filtering by priority keywords

**Code:**

locs = [('Omnicom', 'IN', 'New York'),

('DDB Needham' 'IN' 'New York')

('Kaplan Thaler G'roup'', 'IN', 'New Y' ork'), ('BBDO South' 'IN' 'Atlanta')

('Georgia-Pacif'ic', 'I'N', '

'Atlanta')]

query = [e1 for (e1, rel, e2) in locs if e2=='Atlanta'] print(query)

## Output:

['BBDO South', 'Georgia-Pacific']

## Conclusion:

Thus, in the above experiment we have studied regarding named entity recognition, working of named entity recognition, how named entity recognition can be used and then implemented the code for the same and successfully executed it.

# EXPERIMENT:6

**Aim**: [POS Tagging: Hidden Markov Model](https://nlp-iiith.vlabs.ac.in/exp/markov-model/): experiment is to calculate emission and transition matrix which will be helpful for tagging Parts of Speech using Hidden Markov Model.

## Theory:

It is a process of converting a sentence to forms –list of words, list of tuples where each tuple is having a form(word, tag)).The tag in case of is a part-of-speech tag, and signifies whether the word is a noun, adjective, verb, and so on. Default tagging is a basic step for the part-of-speech tagging. It is performed using the Default Tagger class. The Default Tagger class takes ‘tag’ as a single argument. NN is the tag for a singular noun. Default Taggeris most useful when it gets to work with most common part-of-speech tag. that’s why a noun tag is recommended.

Default taggingis a basic step for the part-of-speech tagging. It is performed sing the Default Tagger class. The Default Tagger class takes ‘tag’ as a single argument. NN is the tag for a singular noun.DefaultTaggeris most useful when it gets to work with most common part-of-speech tag. That’s why a noun tag is recommended.

Tagging is a kind of classification that may be defined as the automatic assignment of description to the tokens. Here the descriptor is called tag, which may represent one of the part-of- speech, semantic information and so on.Now, if we talk about Part-of-Speech (PoS) tagging, then it may be defined as the process of assigning one of the parts of speech to the given word. It is generally called POS tagging. In simple words, we can say that POS tagging is a task of labelling each word in a sentence with its appropriate part of speech. We already know that parts of speech include nouns, verb, adverbs, adjectives, pronouns, conjunction and their sub- categories.

## Rule-based POS Tagging

One of the oldest techniques of tagging is 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 canalso 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.

## Stochastic POS Tagging

Another technique of tagging is Stochastic POS Tagging. Now, the question that arises here is which model can be stochastic. 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.The simplest stochastic tagger applies the following approaches for POS tagging –

## Word Frequency Approach

In this approach, the stochastic taggers disambiguate the words based onthe probability that a word occurs with a particular tag. We can also say that the tag encountered most frequently with the word in the training set is the one assigned to an ambiguous instance of that word. The main issue with this approach is that it may yield inadmissible sequence of tags.

## Tag Sequence Probabilities

It is another approach of stochastic tagging, where the tagger calculates the probability of a given sequence of tags occurring. It is also called n-gram approach. It is called so because the best tag for a given word is determined by the probability at which it occurs with the n previous tags.

## Transformation-based Tagging

Transformation based tagging is also called Brill tagging. It is an instance of the transformation- based learning (TBL), which is a rule-based algorithm for automatic tagging of POS to the given text. TBL, allows us to have linguistic knowledge in a readable form, transforms one state to another state by using transformation rules.

It draws the inspiration from both the previous explained taggers − rule-based and stochastic. If we see similarity between rule-based and transformation tagger, then like rule-based, it is also based on the rules that specify what tags need to be assigned to what words. On the other hand, if we see similarity between stochastic and transformation tagger then like stochastic, it is machine learning technique in which rules are automatically induced from data.

## HMM for POS Tagging

The POS tagging process is the process of finding the sequence of tags which is most likely to have generated a given word sequence. We can model this POS process by using a Hidden 16Markov Model (HMM), wheretagsare thehidden statesthat produced the observable output, i.e., the words.

Code:

Import nltk nltk.download('averaged\_perceptron\_tagger') nltk.download('punkt')

text = nltk.word\_tokenize("And now for Everything completely Same") nltk.pos\_tag(text)

Output:

[('And', 'CC'),

('now', 'RB'),

('for', 'IN'),

('Everything', 'VBG'),

('completely', 'RB'),

('Same', 'JJ')]

## Conclusion:

Thus, we have studied POS Tagging in the above experiment also learned regarding different types of POS Tagging and tried to implement the code for POS Tagging and successfully executed it.

# EXPERIMENT NO:7

**Aim:** [POS Tagging: Viterbi Decoding](https://nlp-iiith.vlabs.ac.in/exp/viterbi-decoding/): experiment is to find POS tags of words in a sentence using Viterbi decoding.

## Theory:

The Natural Language Toolkit (NLTK) is a platform used for building programs for text analysis. One of the more powerful aspects of the NLTK module is the Part of Speech tagging. In order to run the below python program you must have to install NLTK. Please follow the installation steps.

* Open your terminal, run **pip install nltk**.
* Write python in the command prompt so python Interactive Shell is ready to execute your code/Script.
* Type **import nltk**

## nltk.download()

A GUI will pop up then choose to download “all” for all packages, and then click ‘download’. This will give you all of the tokenizers, chunkers, other algorithms, and all of the corpora, so that’s why installation will take quite time.

## Examples:

import nltk nltk.download()

let’sk nock out some quick vocabulary: **Corpus:** Body of text, singular. Corpora is the plural of this. **Lexicon:** Words and their meanings.

**Token:** Each “entity” that is a part of whatever was split up based on rules. In corpus linguistics, **part-of-speech tagging** (**POS tagging** or **PoS tagging** or **POST**), also called **grammatical tagging** or **word-category disambiguation**.

Input: Everything is all about money. Output: [('Everything', 'NN'), ('is', 'VBZ'),

('all', 'DT'),('about', 'IN'),

('money', 'NN'), ('.', '.')]

Text may contain stop words like ‘the’, ‘is’, ‘are’. Stop words can be filtered from the text to be processed. There is no universal list of stop words in nlp research, however the nltk module contains a list of stop words. You can add your own Stop word. Go to your NLTK download **directory path** -> **corpora** -

> **stopwords** -> update the stop word **file** depends on your language which one you are using. Here we are using english (stopwords.words(‘english’)).

* **code**

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize, sent\_tokenize stop\_words = set(stopwords.words('english'))

// Dummy text

txt = "Sukanya, Rajib and Naba are my good friends. " \ "Sukanya is getting married next year. " \

"Marriage is a big step in one’s life." \ "It is both exciting and frightening. " \

"But friendship is a sacred bond between people." \ "It is a special kind of love between us. " \

"Many of you must have tried searching for a friend "\ "but never found the right one."

# sent\_tokenize is one of instances of

# PunktSentenceTokenizer from the nltk.tokenize.punkt module

tokenized = sent\_tokenize(txt) for i in tokenized:

# Word tokenizers is used to find the words # and punctuation in a string

wordsList = nltk.word\_tokenize(i)

# removing stop words from wordList

wordsList = [w for w in wordsList if not w in stop\_words]

# Using a Tagger. Which is part-of-speech # tagger or POS-tagger.

tagged = nltk.pos\_tag(wordsList) print(tagged)

## Output:

[('Sukanya', 'NNP'), ('Rajib', 'NNP'), ('Naba', 'NNP'), ('good', 'JJ'), ('friends', 'NNS')]

[('Sukanya', 'NNP'), ('getting', 'VBG'), ('married', 'VBN'), ('next', 'JJ'), ('year', 'NN')]

[('Marriage', 'NN'), ('big', 'JJ'), ('step', 'NN'), ('one', 'CD'), ('’', 'NN'), ('life', 'NN')]

[('It', 'PRP'), ('exciting', 'VBG'), ('frightening', 'VBG')]

[('But', 'CC'), ('friendship', 'NN'), ('sacred', 'VBD'), ('bond', 'NN'), ('people', 'NNS')]

[('It', 'PRP'), ('special', 'JJ'), ('kind', 'NN'), ('love', 'VB'), ('us', 'PRP')]

[('Many', 'JJ'), ('must', 'MD'), ('tried', 'VB'), ('searching', 'VBG'), ('friend', 'NN'),

('never', 'RB'), ('found', 'VBD'), ('right', 'RB'), ('one', 'CD')]

## Conclusion:

Basically, the goal of a POS tagger is to assign linguistic (mostly grammatical) information to sub-sentential units. Such units are called tokens and, most of the time, correspond to words and symbols(e.g.punctuation)**.**

# EXPERIMENT:8

**Aim**: [Building POS Tagger](https://nlp-iiith.vlabs.ac.in/exp/pos-tagger/): experiment is to know the importance of context and size of training corpus in learning Parts of Speech.

## Theory:

To start, let us analyze a little about sentence composition. Have you ever stopped to think how we structure phrases? They are not random choices of words — you actually follow a structure when reasoning to make your phrase.Of course, we follow cultural conventions learned from childhood, which may vary a little depending on region or background (you might have noticed, for example, that I use a somewhat ‘weird’ style in my phrasing — that’s because even though I’ve read and learned some english, portuguese is still my mother language and the language that I think in). syntax […] is the set of rules, principles, and processes that govern the structure of sentences (sentence structure) in a given language, usually including word order.

To better be able to depict these rules, it was defined that words belong to classes according to the role that they assume in the phrase. These roles are the things called **“parts of speech”**. Now, the number of distinct roles may vary from school to school, however, there are eight classes (controversies!!) that are generally accepted (for English). In alphabetical listing:

Code:

|  |
| --- |
| Importsys |
| from .core.structures import Document |
| from .preprocessing.tagging import MLTagger |
| from .preprocessing.stemming import PorterStemmer |
| def process(raw\_document, pipeline = ['sentencize','pos','stemming']): |
| document = raw\_document |
| if 'sentencize' in pipeline: |
| document = Document(document) |
| if 'pos' in pipeline: |
| tagger = MLTagger() |
| if isinstance(document, Document): |
| sentences = [tagger.tag(sentence) for sentence in document] |

|  |
| --- |
| document.sentences = sentences |
| else: |
| document = tagger.tag(document) |
| if 'stemming' in pipeline: |
| stemmer = PorterStemmer() |
| if isinstance(document, Document): |
| for sentence in document.sentences: |
| for token\_idx in range(len(sentence)): |
| sentence[token\_idx].repr = stemmer.stem(sentence[token\_idx]) |
| else: |
| for token\_idx in range(len(document)): |
| document[token\_idx] = stemmer.stem(sentence[token\_idx]) |
| return document |

## Lets run it:

$python3

>>>import NLPTools

>>>doc = NLPTools.process("Peter is a funny person, he always eats cabbages with sugar.")

>>>for sentence in doc.sentences:

... for token in sentence.tokens:

... print("("+token.raw+", "+str(token.PoS)+")", end=" ")

out: (<SOS>, None) (pet, NNP) (i, VBZ) (a, DT) (funni, JJ) (person, NN) (,, ,) (he, PRP) (alwai, RB) (eat, VBZ) (cabbag, NNS) (with, IN) (sugar, NN) (<EOS>, None)

What happened? Well, we’re getting the results from the stemmer (its on by default in the pipeline). But we can change it:

>>>doc = NLPTools.process("Peter is a funny person, he always eats cabbages with sugar.", pipeline=['sentencize','pos'])

>>>for sentence in doc.sentences:

... for token in sentence.tokens:

... print("("+token.raw+", "+str(token.PoS)+")", end=" ")

out: (<SOS>, None) (Peter, NNP) (is, VBZ) (a, DT) (funny, JJ) (person, NN) (,, ,) (he, PRP) (always, RB) (eats, VBZ) (cabbages, NNS) (with, IN) (sugar, NN) (<EOS>, None)

**Conclusion**: if you want PoS tagging to work, always do it before stemming. Otherwise failure awaits (since our pipeline is hardcoded, this won’t happen, but the warning remains)!

# EXPERIMENT:9

**Aim:** [Chunking](https://nlp-iiith.vlabs.ac.in/exp/chunking/): experiment is to understand the concept of chunking and get familiar with the basic chunk tagset.

## Theory:

**Chunk extraction or partial parsing** is a process of meaningful extracting short phrases from the sentence (tagged with Part-of-Speech).Chunks are made up of words and the kinds of words are defined using the part-of-speech tags. One can even define a pattern or words that can't be a part of chuck and such words are known as **chinks.** A ChunkRule class specifies what words or patterns to include and exclude in a chunk.

## Defining Chunk patterns:

Chuck patterns are normal regular expressions which are modified and designed to match the part-of-speech tag designed to match sequences of part-of-speech tags. Angle brackets are used to specify an indiviual tag for example - *to match a noun tag.* One can define multiple tags in the same way.

Chunking is a process of extracting plu·ases from unstructured text. Instead of just simple tokens which may not represent the actual meaning of the text, its advisable to use phrases such as **"South Africa"** as a single word instead of **'South'** and **'Africa'** separate words.

Chunking in NLP is Changing a perception by moving a "chunk", or a group of bits of infom1ation, in the direction of a Deductive or Inductive conclusion tlu·ough the use of language. Chunking up or down allows the speaker to use certain language patterns, to utilize the natural internal process tlu·ough language, to reach for higher meanings or search for more specific bits/portions of missing infonnation.

When we "Chunk Up" the language gets more abstract and there are more chances for

agreement, and when we "Chunk Down" we tend to be looking for the specific details that may have been missing in the chunk up.

As an example, if you ask the question "for what purpose cars?" you may get the answer "transport", which is a higher chunk and more toward abstract.

If you asked "what specifically about a car"? you will start to get smaller pieces of infom1ation about a car.Lateral thinking will be the process of chunking up and then looking for other examples: For example, "for what intentions cars?", "transportation", "what are other examples of transportation?""Buses!"

## Code:

**Noun Phrase chunking:**

Import nltk

sentence= [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"), ("dog", "NN"), ("barked",

"VBD"), ("at", "IN"), ("the", "DT"), ("cat", "NN")] grammar= "NP: {<DT>?<JJ>\*<NN>}"

cp = nltk.RegexpParser(grammar) result= cp.parse(sentence) print(result)

>>> result.draw()

**Output:**

(S

(NP the/DT little/JJ yellow/JJ dog/NN) barked/VBD at/IN

(NP the/DT cat/NN))



*(/*

**NP**

VBD at IN

**NP**

OT

**UN**

## Conclusion:

Thus, in the above experiment we have studies regarding chunking and tried to implement the code for same and successfully executed it.

# EXPERIMENT:10

**Aim:** [Building Chunker](https://nlp-iiith.vlabs.ac.in/exp/building-chunker/): experiment is to know the importance of selecting proper features for training a model and size of training corpus in learning how to do chunking.

## Theory:

These tools can be very helpful for kids who struggle with writing.To use word prediction, your child needs to use a keyboard to write. This can be an onscreen keyboard on a smartphone or digital tablet. Or it can be a physical keyboard connected to a device or computer.

Those suggestions are shown on the screen, like at the top of an onscreen keyboard. The child clicks or taps on a suggested word, and it's inserted into the writing.

There are also advanced word prediction tools available. They include:

Tools that read word choices aloud with text-to-speech. This is important for kids with reading issues who can't read what the suggestions are.

Word prediction tools that make suggestions tailored to specific topics. For instance, the words used in a history paper will differ a lot from those in a science report. To make suggestions more accurate, kids can pick special dictionaries for what they're writing about.

Tools that display word suggestions in example sentences. This can help kids decide between words that are confusing, like to, too and two.

## Code:

import nltk

from nltk.corpus import state\_union

from nltk.tokenize import PunktSentenceTokenizer

train\_text = state\_union.raw("2005-GWBush.txt") sample\_text = state\_union.raw("2006-GWBush.txt")

custom\_sent\_tokenizer = PunktSentenceTokenizer(train\_text)

tokenized = custom\_sent\_tokenizer.tokenize(sample\_text) def process\_content():

try:

for i in tokenized:

words = nltk.word\_tokenize(i) tagged = nltk.pos\_tag(words)

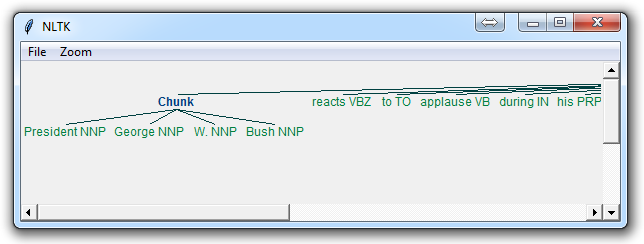
chunkGram = r"""Chunk: {<RB.?>\*<VB.?>\*<NNP>+<NN>?}""" chunkParser = nltk.RegexpParser(chunkGram)

chunked = chunkParser.parse(tagged) chunked.draw()

except Exception as e: print(str(e))

process\_content()

**output:**



**Conclusion:** Tools that display word suggestions in example sentences. This can help kids decide between words that are confusing, like to, too and two.

)