Natural Language Processing

Journal

M.Sc. I.T Part II Semester IV

2020-2021

RJSPGIT4P1

Name: Jitendra Sharma
Roll No: 12

Hindi Vidya Prachar Samiti's

RAMNIRANJAN JHUNJHUNWALA COLLEGE

(AUTONOMOUS)

Opposite Ghatkopar Railway Station, Ghatkopar West, Mumbai-400086

CERTIFICATE

This is to certify that **Mr. Jitendra Sharma** with Seat No.**12** has successfully completed the necessary course of experiments in the subject of **NATURAL LANGUAGE PROCESSING** during the academic year **2020 – 2021** complying with the requirements of **RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE AND COMMERCE**, for the course of **M.Sc. (IT)** semester -IV.

Internal Examiner Date:

Head of Department College Seal External Examiner

Index

SR. NO.	PRACTICAL TITLE	DATE	Page No.
1.	**Regular Expression**		
2.	**Processing Raw Text** 1. Accessing text from Web 2. Accessing local text file 3. Accessing text from PDF, Word and other Binary Format 4. The NLP pipeline		
3.	**Text Processing – 1** Tokenize Text using NLTK 1. Sentence Tokenization 2. PunktSentenceTokenizer 3. Tokenize sentence of different language 4. Word Tokenization 5. Using TreebankWordTokenizer 6. PunktWordTokenizer 7. WordPunctTokenizer 8. Using Regular Expression		
4.	**Text Processing – 2** 1. Accessing Text Corpora a. Gutenberg Corpus b. Web and Chat Text c. Brown Corpus d. Annotated Text Corpora e. Corpora in other languages f. Text Corpus Structure g. Loading your own corpus 2. Word Counting 3. Word Vocabulary 4. Bag-of-words, BOW Model 5. TF/IDF Vectorizer 6. Tokenisation and Word Frequencies 7. Sentence Segmentation 8. Removing Stop Words with NLTK		

5.	**Text Processing** 1. Text Lowercase 2. Remove Numbers 3. Remove Punctuation 4. Remove Whitespaces 5. Remove Default Stopwords 6. Stemming 7. Lemmatization		
6.	**Phonetic Hashing Using Soundex Algorithm**		
7.	**Information Extraction** 1. Part-of-Speech Tagging 2. Chunking 3. Chinking 4. Named Entity Recognition 5. Relation Extraction		
8.	**Classification** 1. Supervised Classification - Gender Identification 2. Choosing right features 3. Document classification 4. Sentence Segmentation 5. Naive Bayes Classifier		
	Parsing Tree POS Tagging		
	Using Wordnet Finding Synonym and Antonym		
	Word Sense Disambiguation - Leak Algorithm		
	Word2Vec		

Practical No. 1: Regular Expression

Aim: Write a program for regular expression

Description:

To the Python interpreter, a regular expression is just like any other string. If the string contains a backslash followed by particular characters, it will interpret these specially. For example \b would be interpreted as the backspace character. In general, when using regular expressions containing backslash, we should instruct the interpreter not to look inside the string at all, but simply to pass it directly to the re library for processing. We do this by prefixing the string with the letter r, to indicate that it is a raw string. For example, the raw string r'\band\b' contains two \b symbols that are interpreted by the re library as matching word boundaries instead of backspace characters. If you get into the habit of using r'...' for regular expressions — as we will do from now on — you will avoid having to think about these complications.

Code:

```
>> import re, nltk
>> from nltk import word_tokenize
>> nltk.download('words')
>> wordlist = [w for w in nltk.corpus.words.words('en') if w.islower()]
```

Let's find words ending with ed using the regular expression «ed$»

```
>> [w for w in wordlist if re.search('ed$', w)]
>> [w for w in wordlist if re.search('^..j..t..$', w)]
>> [w for w in wordlist if re.search('^[ghi][mno][jlk][def]$', w)]
>> nltk.download('nps_chat')
>> chat_words = sorted(set(w for w in nltk.corpus.nps_chat.words()))
>> [w for w in chat_words if re.search('^m+i+n+e+$', w)]
>> [w for w in chat_words if re.search('^[ha]+$', w)]
>> nltk.download('toolbox')
>> rotokas_words = nltk.corpus.toolbox.words('rotokas.dic')
>>
cvs = [cv for w in rotokas_words for cv in re.findall(r'[ptksvr][aeiou]', w)]
>> cfd = nltk.ConditionalFreqDist(cvs)
>> cfd.tabulate()
```

Output:

Regular_Expression_01_Jitendra_Sharma_12.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

Jitendra_Sharma - Roll No. - 12

Practical - 01

Regular Expression

```
[1] import re, nltk
```

```
[2] from nltk import word_tokenize
```

```
[4] nltk.download('words')
```

```
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
True
```

```
[5] wordlist = [w for w in nltk.corpus.words.words('en') if w.islower()]
```

Let's find words ending with ed using the regular expression «ed$»

Regular_Expression_01_Jitendra_Sharma_12.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

Let's find words ending with ed using the regular expression «ed$»

```
[w for w in wordlist if re.search('ed$', w)]
```

```
['abaissed',
 'abandoned',
 'abased',
 'abashed',
 'abatised',
 'abed',
 'aborted',
 'abridged',
 'abscessed',
 'absconded',
 'absorbed',
 'abstracted',
 'abstricted',
 'accelerated',
 'accepted',
 'accidented',
 'accoladed',
 'accolated',
 'accomplished',
 'accosted',
 'accredited',
 'accursed',
```

Regular_Expression_01_Jitendra_Sharma_12.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
 'angled',
```

```
[ ] [w for w in wordlist if re.search('^..j..t..$', w)]
```

```
['abjectly',
 'adjuster',
 'dejected',
 'dejectly',
 'injector',
 'majestic',
 'objectee',
 'objector',
 'rejecter',
 'rejector',
 'unjilted',
 'unjolted',
 'unjustly']
```

```
[ ] [w for w in wordlist if re.search('^[ghi][mno][jlk][def]$', w)]
```

```
['gold', 'golf', 'hold', 'hole']
```

```
[ ] nltk.download('nps_chat')
```

```
[nltk_data] Downloading package nps_chat to /root/nltk_data...
[nltk_data]   Unzipping corpora/nps_chat.zip.
```

✓ 0s completed at 9:08 PM

Regular_Expression_01_Jitendra_Sharma_12.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```python
chat_words = sorted(set(w for w in nltk.corpus.nps_chat.words()))
```

```python
[w for w in chat_words if re.search('^m+i+n+e+$', w)]
```

```
['miiiiiiiiiiiiinnnnnnnnnnneeeeeeeeee',
 'miiiiiinnnnnnnnnneeeeeeee',
 'mine',
 'mmmmmmmmiiiiiiiiinnnnnnnnneeeeeeee']
```

```python
[w for w in chat_words if re.search('^[ha]+$', w)]
```

```
['a',
 'aaaaaaaaaaaaaaaaa',
 'aaahhhh',
 'ah',
 'ahah',
 'ahahah',
 'ahh',
 'ahhahahaha',
 'ahhh',
 'ahhhh',
 'ahhhhhh',
 'ahhhhhhhhhhhhhh',
 'h',
 'ha',
 'haaa',
 'hah'
```

✓ 0s comp

Regular_Expression_01_Jitendra_Sharma_12.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
 'hahhahahah',
 'hahhahahaha']
```

```
[ ] nltk.download('toolbox')
```

```
[nltk_data] Downloading package toolbox to /root/nltk_data...
[nltk_data]   Unzipping corpora/toolbox.zip.
True
```

```
rotokas_words = nltk.corpus.toolbox.words('rotokas.dic')
cvs = [cv for w in rotokas_words for cv in re.findall(r'[ptksvr][aeiou]', w)]
cfd = nltk.ConditionalFreqDist(cvs)
cfd.tabulate()
```

```
    a   e   i   o   u
k 418 148  94 420 173
p  83  31 105  34  51
r 187  63  84  89  79
s   0   0 100   2   1
t  47   8   0 148  37
v  93  27 105  48  49
```

Practical No. 2: Processing Raw Text

Aim: Write a program for processing raw text.

Description:

The most important source of texts is undoubtedly the Web. It's convenient to have existing text collections to explore, such as the corpora we saw in the previous chapters. However, you probably have your own text sources in mind, and need to learn how to access them.

The goal of this chapter is to answer the following questions:

1. How can we write programs to access text from local files and from the web, in order to get hold of an unlimited range of language material?
2. How can we split documents up into individual words and punctuation symbols, so we can carry out the same kinds of analysis we did with text corpora in earlier chapters?
3. How can we write programs to produce formatted output and save it in a file?

In order to address these questions, we will be covering key concepts in NLP, including tokenization and stemming. Along the way you will consolidate your Python knowledge and learn about strings, files, and regular expressions. Since so much text on the web is in HTML format, we will also see how to dispense with markup.

1. Accessing text from Web

A small sample of texts from Project Gutenberg appears in the NLTK corpus collection. However, you may be interested in analyzing other texts from Project Gutenberg. You can browse the catalog of 25,000 free online books at http://www.gutenberg.org/catalog/, and obtain a URL to an ASCII text file. Although 90% of the texts in Project Gutenberg are in English, it includes material in over 50 other languages, including Catalan, Chinese, Dutch, Finnish, French, German, Italian, Portuguese and Spanish (with more than 100 texts each).

Code:

```
>>import nltk, re, pprint
>>from nltk import word_tokenize
>>from urllib import request
>>url = "http://www.gutenberg.org/files/2554/2554-0.txt"
>>response = request.urlopen(url)
>>raw = response.read().decode('utf8')
>>type(raw)
str
>>raw[:75]
The Project Gutenberg EBook of Crime and Punishment, by Fyodor
Dostoevsky
>>import nltk
>>nltk.download('punkt')
>>tokens = word_tokenize(raw)
>>type(tokens)
list
>>len(tokens)
257727
```

```
>>tokens[:10]
['\ufeffThe','Project','Gutenberg','EBook','of','Crime','and','Punishme
nt,
 ',','by']
>>text = nltk.Text(tokens)
>>type(text)
>>text[1024:1062]
['an','exceptionally','hot','evening','early','in','July','a','young','
man','came','out','of','the','garret','in','which','he','lodged','in','
S.','Place','and','walked','slowly',',','as','though','in','hesitation'
,',','towards','K.','bridge','.','He','had','successfully']
>>nltk.download('stopwords')
>>text.collocations()
Katerina Ivanovna; Pyotr Petrovitch; Pulcheria Alexandrovna; Avdotya
Romanovna; Rodion Romanovitch; Marfa Petrovna; Sofya Semyonovna; old
woman; Project Gutenberg-tm; Porfiry Petrovitch; Amalia Ivanovna;
great deal; young man; Nikodim Fomitch; Ilya Petrovitch; Project
Gutenberg; Andrey Semyonovitch; Hay Market; Dmitri Prokofitch; Good
heavens
>>raw.find("PART I")
5336
>>raw.rfind("End of Project Gutenberg's Crime")
-1
>>raw = raw[5338:1157743]
>>raw.find("PART I")
195769
```

2. Accessing local text file

In order to read a local file, we need to use Python's built-in open() function, followed by the read() method. Suppose you have a file document.txt, you can load its contents like this

Note: Your Turn: Create a file called document.txt using a text editor, and type in a few lines of text, and save it as plain text. If you are using IDLE, select the *New Window* command in the *File* menu, typing the required text into this window, and then saving the file as document.txt inside the directory that IDLE offers in the pop-up dialogue box. Next, in the Python interpreter, open the file using f = open('document.txt'), then inspect its contents using print(f.read()).

Code:

```
>>f = open('document.txt')

>>raw = f.read()

>>f = open('document.txt')

>>import os

>>os.listdir('.')

['.config', 'document.txt', 'sample_data']
```

```
>>f.read()
```

```
f.read()

'Natural Language Processing, or NLP for short, is broadly defined as the automatic manipulation of natural language, like speech and text, by software.\n\nThe study of natural lang
uage processing has been around for more than 50 years and grew out of the field of linguistics with the rise of computers.\n\nIn this post, you will discover what natural language
processing is and why it is so important.\n\nAfter reading this post, you will know:\n\nWhat natural language is and how it is different from other types of data.\nWhat makes workin
g with natural language so challenging.\nWhere the field of NLP came from and how it is defined by modern practitioners.\nKick-start your project with my new book Deep Learning for
Natural Language Processing, including step-by-step tutorials and the Python source code files for all examples.\nNLP uses perceptual, behavioral, and communication techniques to ma
ke it easier for people to change their thoughts and actions.\n\nNLP relies on language processi...'
```

```
>>f = open('document.txt', 'rU')

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1:
DeprecationWarning: 'U' mode is deprecated
  """Entry point for launching an IPython kernel.

>>for line in f:

...     print(line.strip())
```

```
for line in f:
...     print(line.strip())

Natural Language Processing, or NLP for short, is broadly defined as the automatic man:

The study of natural language processing has been around for more than 50 years and gre

In this post, you will discover what natural language processing is and why it is so ir

After reading this post, you will know:

What natural language is and how it is different from other types of data.
What makes working with natural language so challenging.
Where the field of NLP came from and how it is defined by modern practitioners.
Kick-start your project with my new book Deep Learning for Natural Language Processing
NLP uses perceptual, behavioral, and communication techniques to make it easier for pe

NLP relies on language processing but should not be confused with natural language pro

NLP was developed by Richard Bandler and John Grinder, who believed it was possible to

Despite a lack of empirical evidence to support it, Bandler and Grinder published two
```

```
>>nltk.download('gutenberg')

>>path = nltk.data.find('corpora/gutenberg/melville-
moby_dick.txt')

>>raw = open(path, 'rU').read()
```

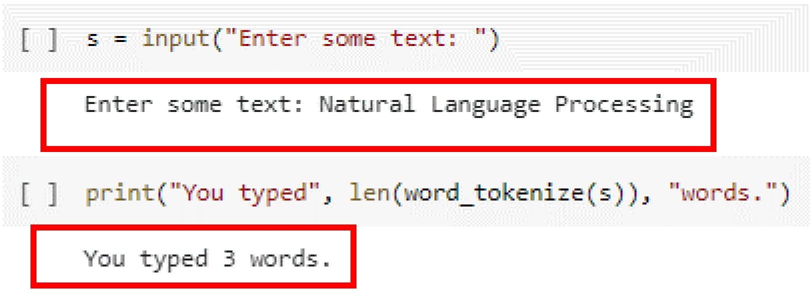
```
raw = open(path, 'rU').read()

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWarning: 'U' mode is deprecated
  """Entry point for launching an IPython kernel.
```

3. Accessing text from PDF, Word and other Binary Format

ASCII text and HTML text are human readable formats. Text often comes in binary formats — like PDF and MSWord — that can only be opened using specialized software. Third-party libraries such as pypdf and pywin32 provide access to these formats. Extracting text from multi-column documents is particularly challenging. For once-off conversion of a few documents, it is simpler to open the document with a suitable application, then save it as text to your local drive, and access it as described below. If the document is already on the web, you can enter its URL in Google's search box. The search result often includes a link to an HTML version of the document, which you can save as text.

Code:

```
>> s = input("Enter some text: ")
>>print("You typed", len(word_tokenize(s)), "words.")
```



4. The NLP pipeline

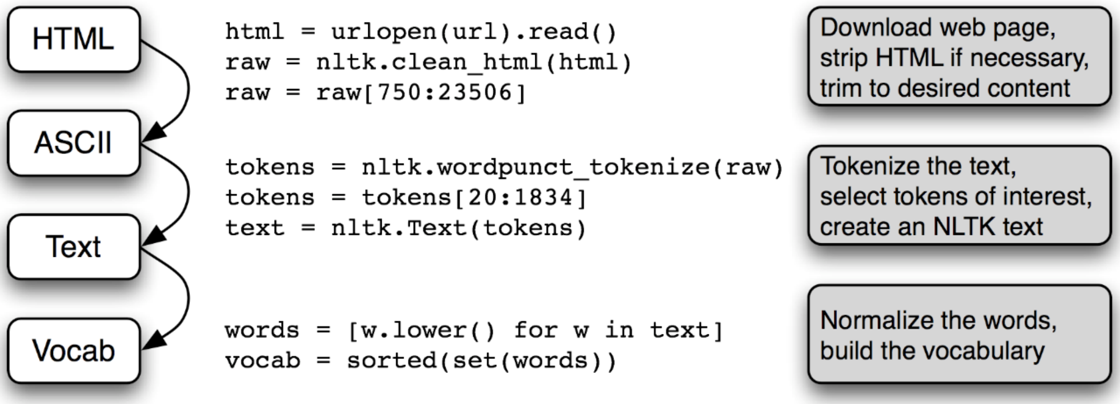


Figure: The Processing Pipeline: We open a URL and read its HTML content, remove the markup and select a slice of characters; this is then tokenized and optionally converted into an nltk.Text object; we can also lowercase all the words and extract the vocabulary. There's a lot going on in this pipeline. To understand it properly, it helps to be clear about the type of each variable that it mentions. We find out the type of any Python object x using type(x), e.g. type(1) is <int> since 1 is an integer.

Code:

```
>>raw = open('document.txt').read()
>>type(raw)
str
>>tokens = word_tokenize(raw)
>>type(tokens)
list
>>words = [w.lower() for w in tokens]
>>type(words)
```

```
>>vocab = sorted(set(words))

>>type(vocab)
```

list

```
>>vocab.append('blog')

>>print(vocab)

>>print(raw)

>>raw.append('blog')
```

```
raw.append('blog')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-71-8e056f716da5> in <module>()
----> 1 raw.append('blog')

AttributeError: 'str' object has no attribute 'append'
```

SEARCH STACK OVERFLOW

```
>>query = 'Who knows?'

>>beatles = ['john', 'paul', 'george', 'ringo']

>>query + beatles
```

```
query + beatles
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-74-13a24f92f7f5> in <module>()
----> 1 query + beatles

TypeError: can only concatenate str (not "list") to str
```

Practical No. 3: Text Processing – 1

Aim: Write a program for text processing 1

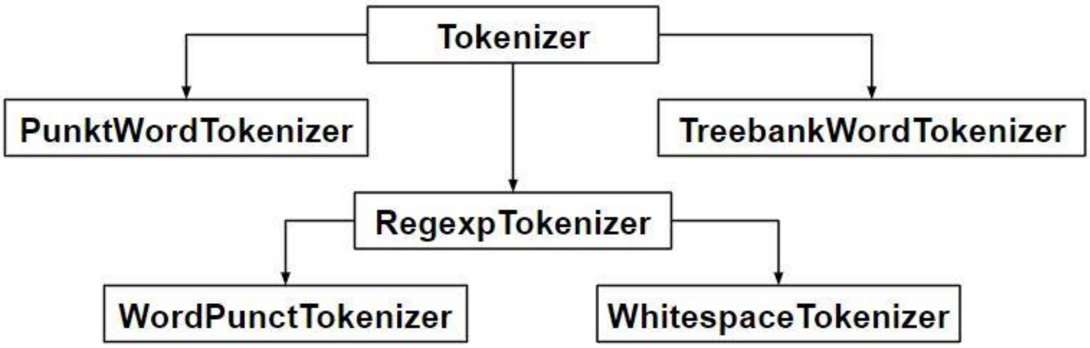
Description:

Natural Language Processing (NLP) is a subfield of computer science, artificial intelligence, information engineering, and human-computer interaction. This field focuses on how to program computers to process and analyze large amounts of natural language data. It is difficult to perform as the process of reading and understanding languages is far more complex than it seems at first glance.

Tokenization is the process of tokenizing or splitting a string, text into a list of tokens. One can think of token as parts like a word is a token in a sentence, and a sentence is a token in a paragraph.

Key points of the article –

- Text into sentences tokenization
- Sentences into words tokenization
- Sentences using regular expressions tokenization



1. **Tokenize Text using NLTK**

To run the below python program, (NLTK) natural language toolkit has to be installed in your system.

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

In order to install NLTK run the following commands in your terminal.

- **sudo pip install nltk**
- Then, enter the python shell in your terminal by simply typing **python**
- Type **import nltk**
- **nltk.download('all')**

Code:

```
# import the existing word and sentence tokenizing
# libraries
>> from nltk.tokenize import sent_tokenize, word_tokenize
>> text = "Natural language processing (NLP) is a field " + \
    "of computer science, artificial intelligence " + \
        "and computational linguistics concerned with " + \
```

```
        "the interactions between computers and human " + \
           "(natural) languages, and, in particular, " + \
           "concerned with programming computers to " + \
           "fruitfully process large natural language " + \
           "corpora. Challenges in natural language " + \
          "processing frequently involve natural " + \
           "language understanding, natural language" + \
           "generation frequently from formal, machine" + \
           "-readable logical forms), connecting language " + \
           "and machine perception, managing human-" + \
           "computer dialog systems, or some combination " + \
           "thereof."
>> import nltk
>> print(sent_tokenize(text))
>> print(word_tokenize(text))
```

```
print(sent_tokenize(text))
print(word_tokenize(text))
```

```
['Natural language processing (NLP) is a field of computer science, artificial intelligence anc
['Natural', 'language', 'processing', '(', 'NLP', ')', 'is', 'a', 'field', 'of', 'computer', 's
```

2. **Sentence Tokenization:** Splitting sentences in the paragraph.

   Code:

```
>> from nltk.tokenize import sent_tokenize
>>text = "Hello everyone. Welcome to GeeksforGeeks. You are studying NL
P article"
>> sent_tokenize(text)
```

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

```
['Hello everyone.',
 'Welcome to GeeksforGeeks.',
 'You are studying NLP article']
```

3. **PunktSentenceTokenizer:** When we have huge chunks of data then it is efficient to use it.

   Code:

```
>> import nltk.data
# Loading PunktSentenceTokenizer using English pickle file
>> tokenizer = nltk.data.load('tokenizers/punkt/PY3/english.pickle')
>> tokenizer.tokenize(text)
```

```
import nltk.data

# Loading PunktSentenceTokenizer using English pickle file
tokenizer = nltk.data.load('tokenizers/punkt/PY3/english.pickle')

tokenizer.tokenize(text)
```

```
['Hello everyone.',
 'Welcome to GeeksforGeeks.',
 'You are studying NLP article']
```

4. **Tokenize sentence of different language:** One can also tokenize sentence from different languages using different pickle file other than English.

   Code:

```
>> import nltk.data
>>spanish_tokenizer = nltk.data.load('tokenizers/punkt/PY3/spanish.pickle')
>> text = 'Hola amigo. Estoy bien.'
>> spanish_tokenizer.tokenize(text)
```

```
import nltk.data

spanish_tokenizer = nltk.data.load('tokenizers/punkt/PY3/spanish.pickle')

text = 'Hola amigo. Estoy bien.'
spanish_tokenizer.tokenize(text)
```

```
['Hola amigo.', 'Estoy bien.']
```

5. **Word Tokenization:** Splitting words in a sentence.

   Code:

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

```
from nltk.tokenize import word_tokenize

text = "Hello everyone. Welcome to GeeksforGeeks."
word_tokenize(text)
```

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

6. **Using TreebankWordTokenizer:** These tokenizers work by separating the words using punctuation and spaces. And as mentioned in the code outputs above, it does not discard the punctuation, allowing a user to decide what to do with the punctuations at the time of pre-processing.

   Code:

```
>> from nltk.tokenize import TreebankWordTokenizer
>> tokenizer = TreebankWordTokenizer()
>> tokenizer.tokenize(text)
```

```
from nltk.tokenize import TreebankWordTokenizer

tokenizer = TreebankWordTokenizer()
tokenizer.tokenize(text)
```

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

7. **PunktWordTokenizer:** It doen't seperates the punctuation from the words.
   Code:

```
#PunktWordTokenizer – It doen't seperates the punctuation from the word
s.
>> from nltk.tokenize import PunktWordTokenizer
>> tokenizer = PunktWordTokenizer()
>> tokenizer.tokenize("Let's see how it's working.")
```

```
#PunktWordTokenizer – It doen't seperates the punctuation from the words.
from nltk.tokenize import PunktWordTokenizer

tokenizer = PunktWordTokenizer()
tokenizer.tokenize("Let's see how it's working.")
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-45-83f41c678104> in <module>()
      1 #PunktWordTokenizer – It doen't seperates the punctuation from the words.
----> 2 from nltk.tokenize import PunktWordTokenizer
      3 
      4 tokenizer = PunktWordTokenizer()
      5 tokenizer.tokenize("Let's see how it's working.")

ImportError: cannot import name 'PunktWordTokenizer' from 'nltk.tokenize' (/usr/loc

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

8. **WordPunctTokenizer:** It seperates the punctuation from the words.

```
#WordPunctTokenizer – It seperates the punctuation from the words.
>> from nltk.tokenize import WordPunctTokenizer
>> tokenizer = WordPunctTokenizer()
>> tokenizer.tokenize("Let's see how it's working.")
```

```
#WordPunctTokenizer – It seperates the punctuation from the words.

from nltk.tokenize import WordPunctTokenizer

tokenizer = WordPunctTokenizer()
tokenizer.tokenize("Let's see how it's working.")
```

```
['Let', "'", 's', 'see', 'how', 'it', "'", 's', 'working', '.']
```

9. Using Regular Expression

```
>> from nltk.tokenize import regexp_tokenize
>> text = "Let's see how it's working."
>> regexp_tokenize(text, "[\w']+")
```

```
from nltk.tokenize import regexp_tokenize

text = "Let's see how it's working."
regexp_tokenize(text, "[\w']+")
```

```
["Let's", 'see', 'how', "it's", 'working']
```

Practical No. 4: Text Processing – 2

Aim: Write a program a text processing - 2

1. Accessing Text Corpora

As just mentioned, a text corpus is a large body of text. Many corpora are designed to contain a careful balance of material in one or more genres. We examined some small text collections in 1., such as the speeches known as the US Presidential Inaugural Addresses. This particular corpus actually contains dozens of individual texts — one per address — but for convenience we glued them end-to-end and treated them as a single text. 1. also used various pre-defined texts that we accessed by typing `from nltk.book import *`. However, since we want to be able to work with other texts, this section examines a variety of text corpora. We'll see how to select individual texts, and how to work with them.

1. Gutenberg Corpus

```
>> import nltk
>> nltk.download('gutenberg')
>> nltk.download('genesis')
>> nltk.download('inaugural')
>> nltk.download('nps_chat')
>> nltk.download('webtext')
>> nltk.download('treebank')
>> nltk.download('punkt')
>> from nltk.book import*
```

```
import nltk
nltk.download('gutenberg')
nltk.download('genesis')
nltk.download('inaugural')
nltk.download('nps_chat')
nltk.download('webtext')
nltk.download('treebank')
nltk.download('punkt')
from nltk.book import*
```

```
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Unzipping corpora/gutenberg.zip.
[nltk_data] Downloading package genesis to /root/nltk_data...
[nltk_data]   Unzipping corpora/genesis.zip.
[nltk_data] Downloading package inaugural to /root/nltk_data...
[nltk_data]   Unzipping corpora/inaugural.zip.
[nltk_data] Downloading package nps_chat to /root/nltk_data...
[nltk_data]   Unzipping corpora/nps_chat.zip.
[nltk_data] Downloading package webtext to /root/nltk_data...
[nltk_data]   Unzipping corpora/webtext.zip.
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
```

```
>> emma = nltk.corpus.gutenberg.words('austen-emma.txt')
>> len(emma)
>> gutenberg.fileids()
```

```
[ ] emma = nltk.corpus.gutenberg.words('austen-emma.txt')
```

```
[ ] len(emma)
```

```
192427
```

```
gutenberg.fileids()
```

```
['austen-emma.txt',
 'austen-persuasion.txt',
 'austen-sense.txt',
 'bible-kjv.txt',
 'blake-poems.txt',
 'bryant-stories.txt',
 'burgess-busterbrown.txt',
 'carroll-alice.txt',
 'chesterton-ball.txt',
 'chesterton-brown.txt',
 'chesterton-thursday.txt',
 'edgeworth-parents.txt',
 'melville-moby_dick.txt',
 'milton-paradise.txt',
 'shakespeare-caesar.txt',
 'shakespeare-hamlet.txt',
 'shakespeare-macbeth.txt',
 'whitman-leaves.txt']
```

```
>> emma = gutenberg.words('austen-emma.txt')
>> for fileid in gutenberg.fileids():
```

```
...      num_chars = len(gutenberg.raw(fileid))
...      num_words = len(gutenberg.words(fileid))
...      num_sents = len(gutenberg.sents(fileid))
...      num_vocab = len(set(w.lower() for w in gutenberg.words(fileid))
)
...      print(round(num_chars/num_words), round(num_words/num_sents), r
ound(num_words/num_vocab), fileid)
```

```
[ ] emma = gutenberg.words('austen-emma.txt')
```

```
for fileid in gutenberg.fileids():
...     num_chars = len(gutenberg.raw(fileid))
...     num_words = len(gutenberg.words(fileid))
...     num_sents = len(gutenberg.sents(fileid))
...     num_vocab = len(set(w.lower() for w in gutenberg.words(fileid)))
...     print(round(num_chars/num_words), round(num_words/num_sents), round(num_words/num_vocab), fileid)
```

```
5 25 26 austen-emma.txt
5 26 17 austen-persuasion.txt
5 28 22 austen-sense.txt
4 34 79 bible-kjv.txt
5 19 5 blake-poems.txt
4 19 14 bryant-stories.txt
4 18 12 burgess-busterbrown.txt
4 20 13 carroll-alice.txt
5 20 12 chesterton-ball.txt
5 23 11 chesterton-brown.txt
5 18 11 chesterton-thursday.txt
4 21 25 edgeworth-parents.txt
5 26 15 melville-moby_dick.txt
5 52 11 milton-paradise.txt
4 12 9 shakespeare-caesar.txt
4 12 8 shakespeare-hamlet.txt
4 12 7 shakespeare-macbeth.txt
5 36 12 whitman-leaves.txt
```

```
>> macbeth_sentences = gutenberg.sents('shakespeare-macbeth.txt')
>> macbeth_sentences
>> macbeth_sentences[1116]
```

```
[ ] macbeth_sentences = gutenberg.sents('shakespeare-macbeth.txt')
    macbeth_sentences

[['[', 'The', 'Tragedie', 'of', 'Macbeth', 'by', 'William', 'Shakespeare', '1603', ']'], ['Actus', 'Primus', '.'], ...]
```

```
macbeth_sentences[1116]

['Double',
 ',',
 'double',
 ',',
 'toile',
 'and',
 'trouble',
 ';',
 'Fire',
 'burne',
 ',',
 'and',
 'Cauldron',
 'bubble']
```

```
>> longest_len = max(len(s) for s in macbeth_sentences)
>> [s for s in macbeth_sentences if len(s) == longest_len]
```

```
longest_len = max(len(s) for s in macbeth_sentences)
[s for s in macbeth_sentences if len(s) == longest_len]

[['Doubtfull',
  'it',
  'stood',
  ',',
  'As',
  'two',
  'spent',
  'Swimmers',
  ',',
  'that',
  'doe',
  'cling',
  'together'
```

2. Web and Chat Text

```
>> from nltk.corpus import webtext
>> for fileid in webtext.fileids():
     print(fileid, webtext.raw(fileid)[:65], '...')
   ...
```

```
from nltk.corpus import webtext
for fileid in webtext.fileids():
    print(fileid, webtext.raw(fileid)[:65], '...')
...

firefox.txt Cookie Manager: "Don't allow sites that set removed cookies to se ...
grail.txt SCENE 1: [wind] [clop clop clop]
KING ARTHUR: Whoa there!  [clop ...
overheard.txt White guy: So, do you have any plans for this evening?
Asian girl ...
pirates.txt PIRATES OF THE CARRIBEAN: DEAD MAN'S CHEST, by Ted Elliott & Terr ...
singles.txt 25 SEXY MALE, seeks attrac older single lady, for discreet encoun ...
wine.txt Lovely delicate, fragrant Rhone wine. Polished leather and strawb ...
Ellipsis
```

```
>> from nltk.corpus import nps_chat
>> chatroom = nps_chat.posts('10-19-20s_706posts.xml')
>> chatroom[123]
```

```
from nltk.corpus import nps_chat
chatroom = nps_chat.posts('10-19-20s_706posts.xml')
chatroom[123]
```

```
['i',
 'do',
 "n't",
 'want',
 'hot',
 'pics',
 'of',
 'a',
 'female',
 ',',
 'I',
 'can',
 'look',
 'in',
 'a',
 'mirror',
 '.']
```

3. Brown Corpus

```
>> from nltk.corpus import brown
>> nltk.download('brown')
>> brown.categories()
```

```
from nltk.corpus import brown
nltk.download('brown')
brown.categories()
```

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
['adventure',
 'belles_lettres',
 'editorial',
 'fiction',
 'government',
 'hobbies',
 'humor',
 'learned',
 'lore',
 'mystery',
 'news',
 'religion',
 'reviews',
 'romance',
 'science_fiction']
```

```
[ ] brown.words(categories='news')

    ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
```

```
[ ] brown.words(fileids=['cg22'])

    ['Does', 'our', 'society', 'have', 'a', 'runaway', ',', ...]
```

```
[ ] brown.sents(categories=['news', 'editorial', 'reviews'])

    [['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investigation', '
```

```
from nltk.corpus import brown
news_text = brown.words(categories='news')
fdist = nltk.FreqDist(w.lower() for w in news_text)
modals = ['can', 'could', 'may', 'might', 'must', 'will']
for m in modals:
  print(m + ':', fdist[m], end=' ')
```

```
can: 94 could: 87 may: 93 might: 38 must: 53 will: 389
```

4. Annotated Text Corpora

5. Corpora in other languages

```
>> nltk.download('cess_esp')
>> nltk.download('floresta')
>> nltk.download('indian')
>> nltk.corpus.cess_esp.words()
>> nltk.download('udhr')

>> nltk.corpus.floresta.words()
>> nltk.corpus.indian.words('hindi.pos')
>> nltk.corpus.udhr.fileids()
```

```
nltk.corpus.floresta.words()

['Um', 'revivalismo', 'refrescante', 'O', '7_e_Meio', ...]
```

```
[ ] nltk.corpus.indian.words('hindi.pos')

    ['पूर्ण', 'प्रतिबंध', 'हटाओ', ':', 'इराक', 'संयुक्त', ...]
```

```
nltk.corpus.udhr.fileids()

['Abkhaz-Cyrillic+Abkh',
 'Abkhaz-UTF8',
 'Achehnese-Latin1',
 'Achuar-Shiwiar-Latin1',
 'Adja-UTF8',
 'Afaan_Oromo_Oromiffa-Latin1',
 'Afrikaans-Latin1',
 'Aguaruna-Latin1',
 'Akuapem_Twi-UTF8',
 'Albanian_Shqip-Latin1',
 'Amahuaca',
```

```
>> nltk.corpus.udhr.words('Javanese-Latin1')[11:]
```

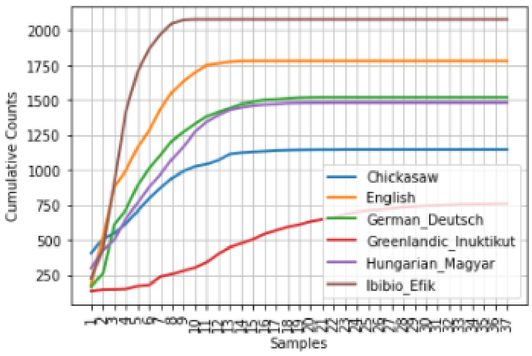
```
>> from nltk.corpus import udhr
>>languages = ['Chickasaw', 'English', 'German_Deutsch','Greenlandic_Inuktikut', 'Hungarian_Magyar', 'Ibibio_Efik']
>> cfd = nltk.ConditionalFreqDist(
...             (lang, len(word))
...             for lang in languages
...             for word in udhr.words(lang + '-Latin1'))
>> cfd.plot(cumulative=True)
```

```
nltk.corpus.udhr.words('Javanese-Latin1')[11:]
```

```
['Saben', 'umat', 'manungsa', 'lair', 'kanthi', 'hak', ...]
```

```
[ ] from nltk.corpus import udhr
    languages = ['Chickasaw', 'English', 'German_Deutsch','Greenlandic_Inuktikut', 'Hungarian_Magyar', 'Ibibio_Efik']
```

```
cfd = nltk.ConditionalFreqDist(
...          (lang, len(word))
...          for lang in languages
...          for word in udhr.words(lang + '-Latin1'))
cfd.plot(cumulative=True)
```



6. Text Corpus Structure

```
>> raw = gutenberg.raw("burgess-busterbrown.txt")
```

```
>> raw[1:20]
>> words = gutenberg.words("burgess-busterbrown.txt")
>> words[1:20]
```

```
raw = gutenberg.raw("burgess-busterbrown.txt")
raw[1:20]
```

```
'The Adventures of B'
```

```
words = gutenberg.words("burgess-busterbrown.txt")
words[1:20]
```

```
['The',
 'Adventures',
 'of',
 'Buster',
 'Bear',
 ...
```

7. **Loading your own corpus**

2. Word Counting

After tokenising a text, the first figure we can calculate is the word frequency. By word frequency we indicate the number of times each token occurs in a text. When talking about word frequency, we distinguish between types and tokens. Types are the distinct words in a corpus, whereas tokens are the words, including repeats. Let's see how this works in practice.

Let's take as example one of the sentences below:

Types are the distinct words in a corpus, whereas tokens are the running words.

How many types and tokens are there in the above sentence? Answer

Let's see how we can use Python to calculate these figures. First of all, let's tokenise the sentence by using a tokeniser which uses non-alphabetical characters as a separator.

Code:

```
>> from nltk.tokenize.regexp import WhitespaceTokenizer
>>my_str = "Types are the distinct words in a corpus, whereas tokens are the running words."
>> tokens = WhitespaceTokenizer().tokenize(my_str)
>> print (len(tokens))
>> print (len(tokens))
>> print (len(tokens))
>>my_str = "Types are the distinct words in a corpus, whereas tokens are the running words."
>> from nltk.tokenize.regexp import WordPunctTokenizer
>> my_toks = WordPunctTokenizer().tokenize(my_str)
>> print (len(my_toks))
>> my_vocab = set(my_toks)
>> print (len(my_vocab))
```

3. Word Vocabulary

The vocabulary serves a few primary purposes:

- help in the preprocessing of the corpus text
- serve as storage location in memory for processed text corpus
- collect and store metadata about the corpus
- allow for pre-task munging, exploration, and experimentation

The vocabulary serves a few related purposes and can be thought of in a few different ways, but the main takeaway is that, once a corpus has made its way to the vocabulary, the text has been processed and any relevant metadata should be collected and stored.

Code:

```
>> sentence = "Thomas Jefferson began building Monticello at the age of 26."
>> sentence.split()
>> str.split(sentence)
```

```
sentence = "Thomas Jefferson began building Monticello at the age of 26."
```

```
[ ] sentence.split()

['Thomas',
 'Jefferson',
 'began',
 'building',
 'Monticello',
 'at',
 'the',
 'age',
 'of',
 '26.']
```

```
str.split(sentence)

['Thomas',
 'Jefferson',
 'began',
 'building',
 'Monticello',
 'at',
 'the',
 'age',
 'of',
 '26.']
```

4. Bag-of-words, BOW Model

The bag-of-words model is a way of representing text data when modeling text with machine learning algorithms.

The bag-of-words model is simple to understand and implement and has seen great success in problems such as language modeling and document classification.

In this tutorial, you will discover the bag-of-words model for feature extraction in natural language processing.

After completing this tutorial, you will know:

- What the bag-of-words model is and why it is needed to represent text.
- How to develop a bag-of-words model for a collection of documents.
- How to use different techniques to prepare a vocabulary and score words.

Code:

```
>> import pandas as pd
>>
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

>> sentence_1="This is a good job.I will not miss it for anything"
>> sentence_2="This is not good at all"
>>CountVec = CountVectorizer(ngram_range=(1,1), # to use bigrams ngram_range=
(2,2)
                            stop_words='english')
#transform
>> Count_data = CountVec.fit_transform([sentence_1,sentence_2])
#create dataframe
>>cv_dataframe=pd.DataFrame(Count_data.toarray(),columns=CountVec.get_feature
_names())
>> print(cv_dataframe)
```

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

sentence_1="This is a good job.I will not miss it for anything"
sentence_2="This is not good at all"



CountVec = CountVectorizer(ngram_range=(1,1), # to use bigrams ngram_range=(2,2)
                           stop_words='english')
#transform
Count_data = CountVec.fit_transform([sentence_1,sentence_2])

#create dataframe
cv_dataframe=pd.DataFrame(Count_data.toarray(),columns=CountVec.get_feature_names())
print(cv_dataframe)
```

```
   good  job  miss
0     1    1     1
1     1    0     0
```

5. TF/IDF Vectorizer

TF-IDF stands for term frequency-inverse document frequency. TF-IDF weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

- **Term Frequency (TF)**: is a scoring of the frequency of the word in the current document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. The term frequency is often divided by the document length to normalize.

$$TF(t) = \frac{\text{Number of times term } t \text{ appears in a document}}{\text{Total number of terms in the document}}$$

- **Inverse Document Frequency (IDF)**: is a scoring of how rare the word is across documents. IDF is a measure of how rare a term is. Rarer the term, more is the IDF score.

$$IDF(t) = log_e\left(\frac{\text{Total number of documents}}{\text{Number of documents with term } t \text{ in it}}\right)$$

Code:

```
>> import pandas as pd
>>
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
>> sentence_1="This is a good job.I will not miss it for anything"
>> sentence_2="This is not good at all"
#without smooth IDF
>> print("Without Smoothing:")
#define tf-idf
>> tf_idf_vec = TfidfVectorizer(use_idf=True,
                        smooth_idf=False,
                        ngram_range=(1,1),stop_words='english') # to use only
  bigrams ngram_range=(2,2)
#transform
>> tf_idf_data = tf_idf_vec.fit_transform([sentence_1,sentence_2])
#create dataframe
>>tf_idf_dataframe=pd.DataFrame(tf_idf_data.toarray(),columns=tf_idf_vec.get_
feature_names())
>> print(tf_idf_dataframe)
>> print("\n")



#with smooth
>> tf_idf_vec_smooth = TfidfVectorizer(use_idf=True,
                        smooth_idf=True,
                        ngram_range=(1,1),stop_words='english')
>>tf_idf_data_smooth = tf_idf_vec_smooth.fit_transform([sentence_1,sentence_2
])
```

```
>> print("With Smoothing:")
>>tf_idf_dataframe_smooth=pd.DataFrame(tf_idf_data_smooth.toarray(),columns=tf_idf_vec_smooth.get_feature_names())
>> print(tf_idf_dataframe_smooth)
```

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

sentence_1="This is a good job.I will not miss it for anything"
sentence_2="This is not good at all"



#without smooth IDF
print("Without Smoothing:")
#define tf-idf
tf_idf_vec = TfidfVectorizer(use_idf=True,
                        smooth_idf=False,
                        ngram_range=(1,1),stop_words='english') # to use only  bigrams ngram_range=(2,2)
#transform
tf_idf_data = tf_idf_vec.fit_transform([sentence_1,sentence_2])

#create dataframe
tf_idf_dataframe=pd.DataFrame(tf_idf_data.toarray(),columns=tf_idf_vec.get_feature_names())
print(tf_idf_dataframe)
print("\n")

#with smooth
tf_idf_vec_smooth = TfidfVectorizer(use_idf=True,
                        smooth_idf=True,
                        ngram_range=(1,1),stop_words='english')


tf_idf_data_smooth = tf_idf_vec_smooth.fit_transform([sentence_1,sentence_2])

print("With Smoothing:")
tf_idf_dataframe_smooth=pd.DataFrame(tf_idf_data_smooth.toarray(),columns=tf_idf_vec_smooth.get_feature_names())
print(tf_idf_dataframe_smooth)
```

```
Without Smoothing:
       good       job      miss
0  0.385372  0.652491  0.652491
1  1.000000  0.000000  0.000000


With Smoothing:
       good       job      miss
0  0.449436  0.631667  0.631667
1  1.000000  0.000000  0.000000
```

6. Tokenisation and Word Frequencies

One of the key steps in NLP or Natural Language Process is the ability to count the frequency of the terms used in a text document or table. To achieve this we must tokenize the words so that they represent individual objects that can be counted. There are a great set of libraries that you can use to tokenize words. However the most popular Python library is NLTK or Natural Language Tool Kit.

Code:

```
>> from bs4 import BeautifulSoup
>> import urllib.request
```

```
>>
page= urllib.request.urlopen('https://statecancerprofiles.cancer.gov/quick-profiles/index.php?statename=newjersey')
```

```
>> soup=BeautifulSoup(page,'html.parser')
>> print(soup)
```

```
from bs4 import BeautifulSoup
import urllib.request
```

```
[ ] page= urllib.request.urlopen('https://statecancerprofiles.cancer.gov/quick-profiles/index.php?statename=newjersey')
```

```
[ ] soup=BeautifulSoup(page,'html.parser')
```

```
print(soup)
```

```
<!DOCTYPE HTML>

<html lang="en">
<head>
<!-- Global site tag (gtag.js) - Google Analytics -->
<script async="" src="https://www.googletagmanager.com/gtag/js?id=UA-112281461-1"></script>
<script>
  window.dataLayer = window.dataLayer || [];
  function gtag(){dataLayer.push(arguments);}
  gtag('js', new Date());
  gtag('config', 'UA-112281461-1');
</script>
```

```
>> import nltk
>> from nltk.tokenize import word_tokenize
#@title Default title text
>> text=soup.get_text(strip=True)
>> print(text)
```

```python
import nltk
from nltk.tokenize import word_tokenize
```

```python
text=soup.get_text(strip=True)
print(text)
```

```
window.dataLayer = window.dataLayer || [];
  function gtag(){dataLayer.push(arguments);}
  gtag('js', new Date());
  gtag('config', 'UA-112281461-1');State Cancer Profiles > Quick ProfilesSkip to
  .notification-banner > div > div {
    padding: 0;
    margin: 0;
  }
  .alert-drawer-wrapper {
    background: #fada87;
    padding: .5em 1.25em;
    margin-bottom: 1px;
  }

  .alert-drawer__body {
    display: none;
  }
  .alert-drawer__body.active {
    display: block;
  }
  .alert-drawer__body.active + .alert-drawer__toggle button::after {
    content: '';
    display: none;
```

```
>> nltk.download('punkt')
>> word_tokens=word_tokenize(text)
>> print(word_tokens)
>> cts=nltk.FreqDist(word_tokens)
>> cts.plot(20)
```

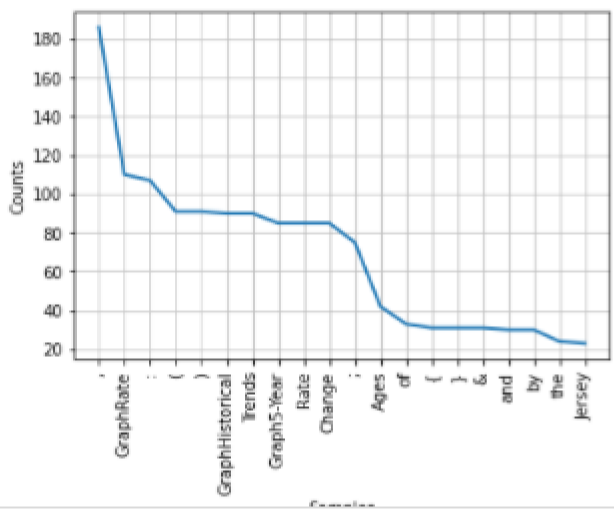
```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
[ ] word_tokens=word_tokenize(text)
    print(word_tokens)
```

```
['window.dataLayer', '=', 'window.dataLayer', '||', '[', ']', ';', 'function', 'gtag', '(', '
```

```
cts=nltk.FreqDist(word_tokens)
cts.plot(20)
```



7. Sentence Segmentation

Sentence tokenization (also called **sentence segmentation**) is the problem of **dividing a string** of written language **into** its component **sentences**. The idea here looks very simple. In English and some other languages, we can split apart the sentences whenever we see a punctuation mark.

Code:

```
#import spacy library
>> import spacy
#load core english library
>> nlp = spacy.load("en_core_web_sm")
#take unicode string
#here u stands for unicode
>>doc = nlp(u"I Love Coding. Geeks for Geeks helped me in this regard very mu
ch. I Love Geeks for Geeks.")
#to print sentences
>> for sent in doc.sents:
>> print(sent)
```

```
#import spacy library
import spacy

#load core english library
nlp = spacy.load("en_core_web_sm")

#take unicode string
#here u stands for unicode
doc = nlp(u"I Love Coding. Geeks for Geeks helped me in this regard very much. I Love Geeks for Geeks.")
#to print sentences
for sent in doc.sents:
  print(sent)
```

```
I Love Coding.
Geeks for Geeks helped me in this regard very much.
I Love Geeks for Geeks.
```

```
>> stanza.download('en')
>> import stanza
>> nlp = stanza.Pipeline(lang='en', processors='tokenize')
>> doc = nlp('This is a test sentence for stanza. This is another sentence.')
>> for i, sentence in enumerate(doc.sentences):
    print(f'====== Sentence {i+1} tokens =======')
    print(*[f'id: {token.id}\ttext: {token.text}' for token in sentence.token
s], sep='\n')
```

```
import stanza

nlp = stanza.Pipeline(lang='en', processors='tokenize')
doc = nlp('This is a test sentence for stanza. This is another sentence.')
for i, sentence in enumerate(doc.sentences):
    print(f'====== Sentence {i+1} tokens =======')
    print(*[f'id: {token.id}\ttext: {token.text}' for token in sentence.tokens], sep='\n')
```

```
2021-04-29 20:03:43 INFO: Loading these models for language: en (English):
========================
| Processor | Package  |
------------------------
| tokenize  | combined |
========================

2021-04-29 20:03:43 INFO: Use device: cpu
2021-04-29 20:03:43 INFO: Loading: tokenize
2021-04-29 20:03:43 INFO: Done loading processors!
====== Sentence 1 tokens =======
id: (1,)        text: This
id: (2,)        text: is
id: (3,)        text: a
id: (4,)        text: test
id: (5,)        text: sentence
id: (6,)        text: for
id: (7,)        text: stanza
id: (8,)        text: .
====== Sentence 2 tokens =======
id: (1,)        text: This
id: (2,)        text: is
id: (3,)        text: another
id: (4,)        text: sentence
id: (5,)        text: .
```

8. Removing Stop Words with NLTK

The NLTK library is one of the oldest and most commonly used Python libraries for Natural Language Processing. NLTK supports stop word removal, and you can find the list of stop words in the corpus module. To remove stop words from a sentence, you can divide your text into words and then remove the word if it exits in the list of stop words provided by NLTK.

Code:

```
>> from nltk.corpus import stopwords
>> nltk.download('stopwords')
>> print(stopwords.words("english"))
#Let's see how we can remove the stop words from a sentence.
>> stop_words = set(stopwords.words("english"))
>> sentence = "Backgammon is one of the oldest known board games."
>> words = nltk.word_tokenize(sentence)
>> without_stop_words = [word for word in words if not word in stop_words]
>> print(without_stop_words)
```

```
from nltk.corpus import stopwords
```

```
[ ] nltk.download('stopwords')
    print(stopwords.words("english"))

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll
```

```
#Let's see how we can remove the stop words from a sentence.
stop_words = set(stopwords.words("english"))
sentence = "Backgammon is one of the oldest known board games."

words = nltk.word_tokenize(sentence)
without_stop_words = [word for word in words if not word in stop_words]
print(without_stop_words)
```

```
['Backgammon', 'one', 'oldest', 'known', 'board', 'games', '.']
```

Practical No. 5: Text Processing

Aim: Write a program to Text Processing.

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.

Code:

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

1. **Text Lowercase :** We lowercase the text to reduce the size of the vocabulary of our text data.

   Code:

   ```
   >> def text_lowercase(text):
   >> return text.lower()
   >>input_str = "Hey, did you know that the summer break is coming? Amazi
   ng >> right !! It's only 5 more days !!"
   >> text_lowercase(input_str)
   ```

2. **Remove Numbers :** We can either remove numbers or convert the numbers into their textual representations.

   We can use regular expressions to remove the numbers.

   ```
   # 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)
```

1. Text Lowercase

```
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)
```

```
'hey, did you know that the summer break is coming? amazing right !! it's only 5 more days !!'
```

2. Remove Numbers

```
# 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)
```

```
'There are  balls in this bag, and  in the other one.'
```

3. **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.
   Code:

```
# 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)
```

4. **Remove Whitespaces:** We can use the join and split function to remove all the white spaces in a string.
   Code:

```
# 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)
```

5. **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.

Code:

```
>> nltk.download('stopwords')
>> nltk.download('punkt')
>> 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_wor
ds]
>> return filtered_text
>>example_text = "This is a sample sentence and we are going to remove
the stopwords from this."
>> remove_stopwords(example_text)
```

```
[5] nltk.download('stopwords')
    nltk.download('punkt')

    [nltk_data] Downloading package stopwords to /root/nltk_data...
    [nltk_data]   Unzipping corpora/stopwords.zip.
    [nltk_data] Downloading package punkt to /root/nltk_data...
    [nltk_data]   Unzipping tokenizers/punkt.zip.
    True
```

```
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(example_text)

['This', 'sample', 'sentence', 'going', 'remove', 'stopwords', '.']
```

6. 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.

Code:

```
>> from nltk.stem.porter import PorterStemmer
>> from nltk.tokenize import word_tokenize
>> stemmer = PorterStemmer()

# stem words in the list of tokenised 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 o
f processes'
>> stem_words(text)
```

```
from nltk.stem.porter import PorterStemmer
from nltk.tokenize import word_tokenize
stemmer = PorterStemmer()

# stem words in the list of tokenised 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)
```

```
['data',
 'scienc',
 'use',
 'scientif',
 'method',
 'algorithm',
 'and',
 'mani',
 'type',
 'of',
 'process']
```

7. **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.

Code:

```
>> nltk.download('wordnet')
>> 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 type
s of processes'
>> lemmatize_word(text)
```

7. Lemmatization

```
[ ] nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
True
```

```
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)
```

```
['data',
 'science',
 'use',
 'scientific',
 'methods',
 'algorithms',
 'and',
 'many',
 'type',
 'of',
 'process']
```

Practical No. 6: Phonetic Hashing Using Soundex Algorithm

Aim: Write a program for Phonetic Hashing Using Soundex Algorithm

Description:

Phonetic hashing buckets all the similar phonemes (words with similar sound or pronunciation) into a single bucket and gives all these variations a single hash code. Hence, the word ‘Dilli’ and ‘Delhi’ will have the same code.

Phonetic hashing is done using the Soundex algorithm. It doesn’t matter which language the input word comes from — as long as the words sound similar, they will get the same hash code.

Now, let’s see it through an example. The Soundex of the word ‘Mississippi’. To calculate the hash code, following are the steps:
Phonetic hashing is a four-letter code. The first letter of the code is the first letter of the input word. Hence it is retained as is. The first character of the phonetic hash is ‘M’. Now, we need to make changes to the rest of the letters of the word.
Now, we need to map all the consonant letters (except the first letter). All the vowels are written as is and ‘H’s, ‘Y’s and ‘W’s remain unencoded (unencoded means they are removed from the word). After mapping the consonants, the code becomes MI22I22I11I.
The third step is to remove all the vowels. ‘I’ is the only vowel. After removing all the ‘I’s, we get the code M222211. Now, you would need to merge all the consecutive duplicate numbers into a single unique number. All the ‘2’s are merged into a single ‘2’. Similarly, all the ‘1’s are merged into a single ‘1’. The code that we get is M21.
The fourth step is to force the code to make it a four-letter code. You either need to pad it with zeroes in case it is less than four characters in length. Or you need to truncate it from the right side in case it is more than four characters in length. Since the code is less than four characters in length, you’ll pad it with one ‘0’ at the end. The final code is M210.

Code:

```
>> import numpy as np
>> import pandas as pd

#Visualization Libraries
>> import seaborn as sns
>> import matplotlib.pyplot as plt


#imports from sklearn library
>> from sklearn import datasets
>> from sklearn.linear_model import LinearRegression
>>
from sklearn.model_selection import train_test_split, cross_val_score
>> from sklearn.metrics import mean_squared_error


#loading the dataset direclty from sklearn
>> boston = datasets.load_boston()
```

```
>> print('\n')
>> print(boston.keys())
>> print('\n')
>> print(boston.data.shape)
>> print('\n')
>> print(boston.feature_names)
```

```
print('\n')
print(boston.keys())
print('\n')
print(boston.data.shape)
print('\n')
print(boston.feature_names)
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])


(506, 13)


['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

```
>> print(boston.DESCR)
```

```
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's
```

```
>> bos = pd.DataFrame(boston.data, columns = boston.feature_names)
>> bos['PRICE'] = boston.target

>> print(bos.head())
```

```
[ ]  bos = pd.DataFrame(boston.data, columns = boston.feature_names)
     bos['PRICE'] = boston.target

     print(bos.head())
```

```
      CRIM    ZN  INDUS  CHAS    NOX  ...    TAX  PTRATIO       B  LSTAT  PRICE
0  0.00632  18.0   2.31   0.0  0.538  ...  296.0     15.3  396.90   4.98   24.0
1  0.02731   0.0   7.07   0.0  0.469  ...  242.0     17.8  396.90   9.14   21.6
2  0.02729   0.0   7.07   0.0  0.469  ...  242.0     17.8  392.83   4.03   34.7
3  0.03237   0.0   2.18   0.0  0.458  ...  222.0     18.7  394.63   2.94   33.4
4  0.06905   0.0   2.18   0.0  0.458  ...  222.0     18.7  396.90   5.33   36.2

[5 rows x 14 columns]
```

```
>> bos.isnull().sum()
```

```
bos.isnull().sum()
```

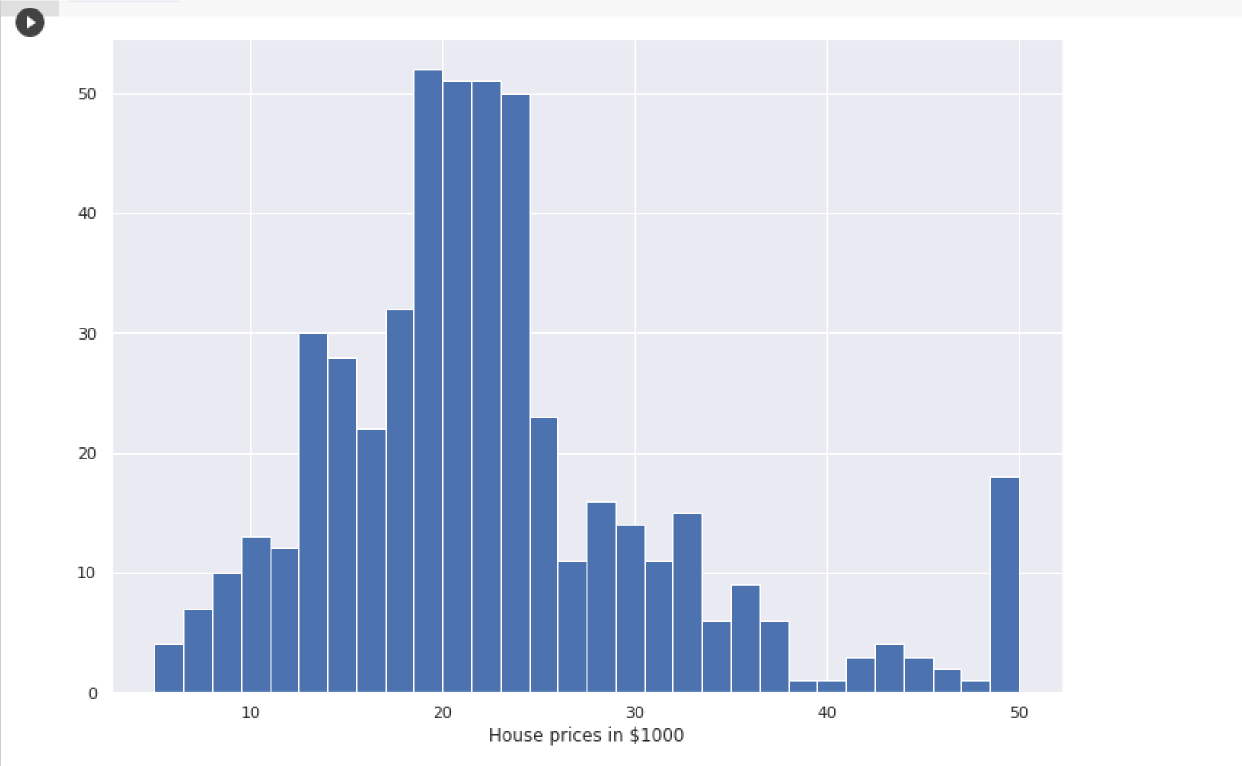
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
PRICE      0
dtype: int64
```

```
>> print(bos.describe())
```

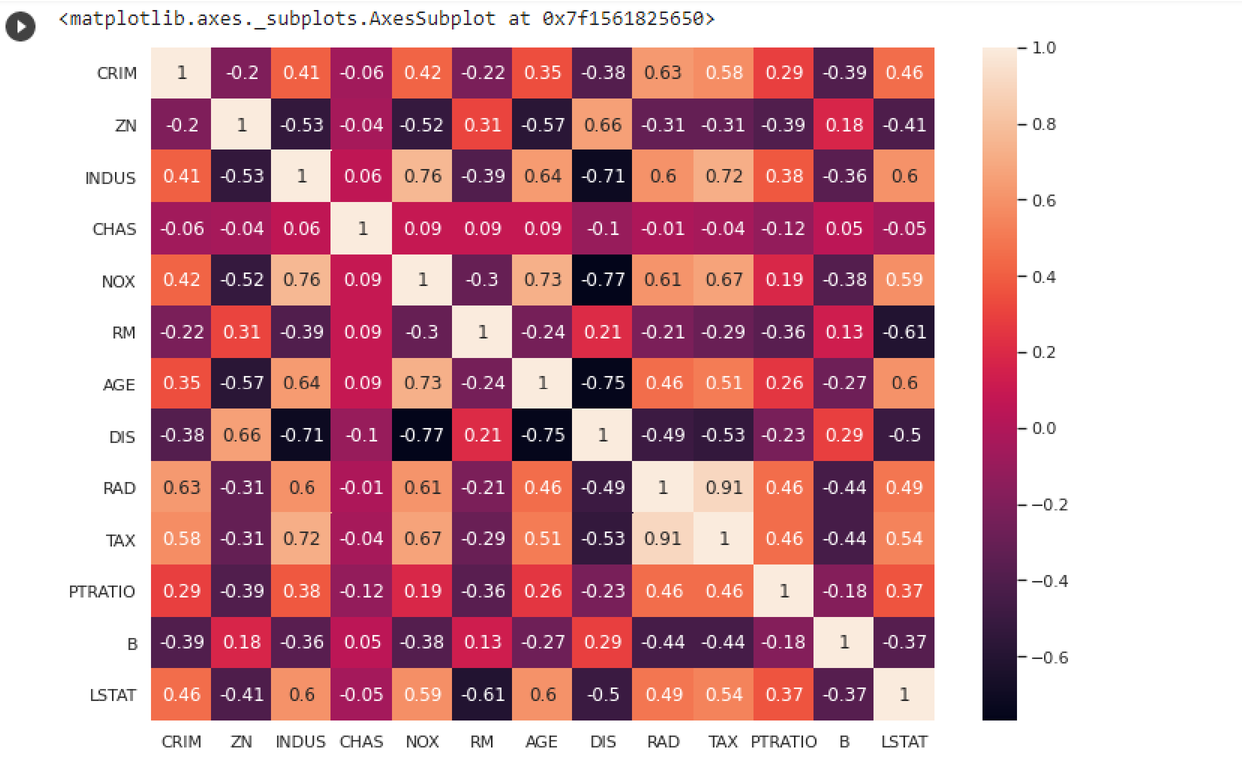
```
print(bos.describe())
```

```
             CRIM          ZN       INDUS  ...           B       LSTAT       PRICE
count  506.000000  506.000000  506.000000  ...  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779  ...  356.674032   12.653063   22.532806
std      8.601545   23.322453    6.860353  ...   91.294864    7.141062    9.197104
min      0.006320    0.000000    0.460000  ...    0.320000    1.730000    5.000000
25%      0.082045    0.000000    5.190000  ...  375.377500    6.950000   17.025000
50%      0.256510    0.000000    9.690000  ...  391.440000   11.360000   21.200000
75%      3.677083   12.500000   18.100000  ...  396.225000   16.955000   25.000000
max     88.976200  100.000000   27.740000  ...  396.900000   37.970000   50.000000

[8 rows x 14 columns]
```

```
>> sns.set(rc={'figure.figsize':(11.7,8.27)})
>> plt.hist(bos['PRICE'], bins=30)
>> plt.xlabel("House prices in $1000")
>> plt.show()
```



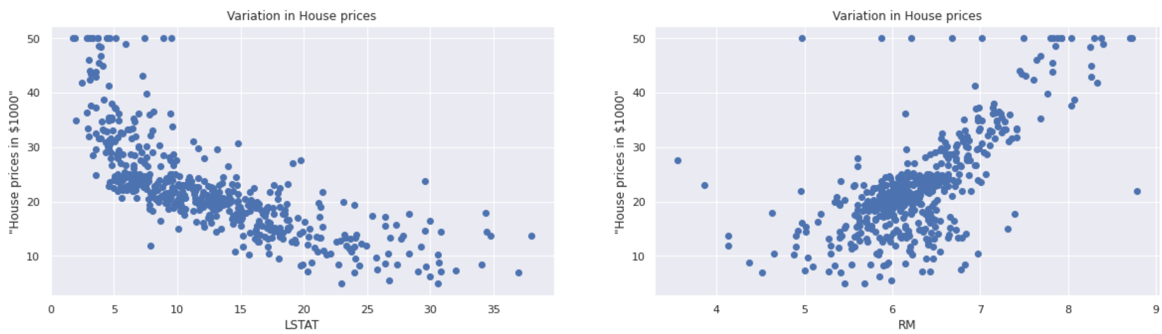

```
#Created a dataframe without the price col, since we need to see the co
rrelation between the variables
>> bos_1 = pd.DataFrame(boston.data, columns = boston.feature_names)

>> correlation_matrix = bos_1.corr().round(2)
>> sns.heatmap(data=correlation_matrix, annot=True)
```




```
>> plt.figure(figsize=(20, 5))

>> features = ['LSTAT', 'RM']
>> target = bos['PRICE']
```

```
>> for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = bos[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title("Variation in House prices")
    plt.xlabel(col)
    plt.ylabel('"House prices in $1000"')
```

```
plt.subplot(1, len(features) , i+1)
x = bos[col]
y = target
plt.scatter(x, y, marker='o')
plt.title("Variation in House prices")
plt.xlabel(col)
plt.ylabel('"House prices in $1000"')
```




```
>> X_rooms = bos.RM
>> y_price = bos.PRICE


>> X_rooms = np.array(X_rooms).reshape(-1,1)
>> y_price = np.array(y_price).reshape(-1,1)

>> print(X_rooms.shape)
>> print(y_price.shape)
```

```
X_rooms = bos.RM
y_price = bos.PRICE


X_rooms = np.array(X_rooms).reshape(-1,1)
y_price = np.array(y_price).reshape(-1,1)

print(X_rooms.shape)
print(y_price.shape)

(506, 1)
(506, 1)
```

```
>>
X_train_1, X_test_1, Y_train_1, Y_test_1 = train_test_split(X_rooms, y_
price, test_size = 0.2, random_state=5)
```

```
>> print(X_train_1.shape)
>> print(X_test_1.shape)
>> print(Y_train_1.shape)
>> print(Y_test_1.shape)
```

```
X_train_1, X_test_1, Y_train_1, Y_test_1 = train_test_split(X_rooms, y_price, test_size = 0.2, random_state=5)

print(X_train_1.shape)
print(X_test_1.shape)
print(Y_train_1.shape)
print(Y_test_1.shape)
```

```
(404, 1)
(102, 1)
(404, 1)
(102, 1)
```

```
>> reg_1 = LinearRegression()
>> reg_1.fit(X_train_1, Y_train_1)

>> y_train_predict_1 = reg_1.predict(X_train_1)
>> rmse = (np.sqrt(mean_squared_error(Y_train_1, y_train_predict_1)))
>> r2 = round(reg_1.score(X_train_1, Y_train_1),2)

>> print("The model performance for training set")
>> print("--------------------------------------")
>> print('RMSE is {}'.format(rmse))
>> print('R2 score is {}'.format(r2))
>> print("\n")
```

```
reg_1 = LinearRegression()
reg_1.fit(X_train_1, Y_train_1)

y_train_predict_1 = reg_1.predict(X_train_1)
rmse = (np.sqrt(mean_squared_error(Y_train_1, y_train_predict_1)))
r2 = round(reg_1.score(X_train_1, Y_train_1),2)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")
```

```
The model performance for training set
--------------------------------------
RMSE is 6.972277149440585
R2 score is 0.43
```

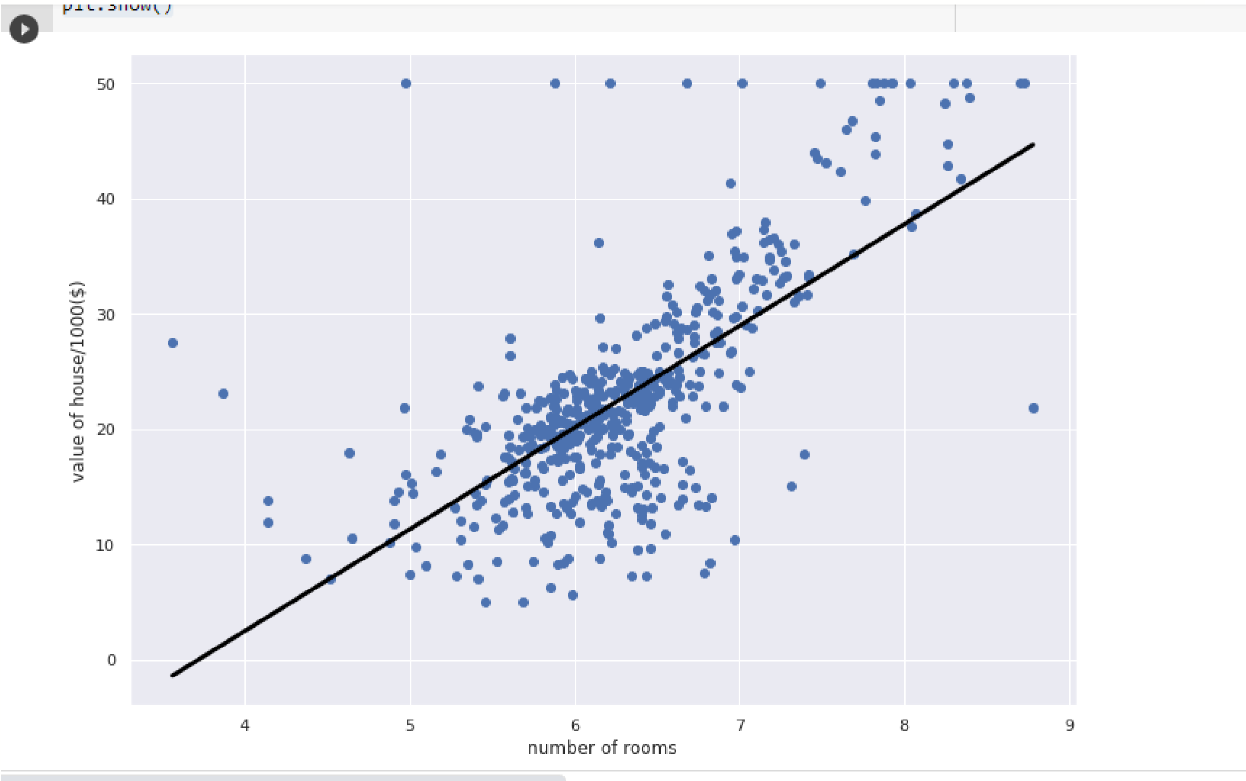
```
>> y_pred_1 = reg_1.predict(X_test_1)
>> rmse = (np.sqrt(mean_squared_error(Y_test_1, y_pred_1)))
>> r2 = round(reg_1.score(X_test_1, Y_test_1),2)

>> print("The model performance for training set")
>> print("--------------------------------------")
>> print("Root Mean Squared Error: {}".format(rmse))
>> print("R^2: {}".format(r2))
```

```
>> print("\n")
```

```
y_pred_1 = reg_1.predict(X_test_1)
rmse = (np.sqrt(mean_squared_error(Y_test_1, y_pred_1)))
r2 = round(reg_1.score(X_test_1, Y_test_1),2)

print("The model performance for training set")
print("--------------------------------------")
print("Root Mean Squared Error: {}".format(rmse))
print("R^2: {}".format(r2))
print("\n")
```

```
The model performance for training set
--------------------------------------
Root Mean Squared Error: 4.895963186952216
R^2: 0.69
```

```
>> prediction_space = np.linspace(min(X_rooms), max(X_rooms)).reshape(-1,1)
>> plt.scatter(X_rooms,y_price)
>>
plt.plot(prediction_space, reg_1.predict(prediction_space), color = 'black', linewidth = 3)
>> plt.ylabel('value of house/1000($)')
>> plt.xlabel('number of rooms')
>> plt.show()
```




```
>> X = bos.drop('PRICE', axis = 1)
>> y = bos['PRICE']
```

```
>>
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,
random_state=42)

>> reg_all = LinearRegression()
>> reg_all.fit(X_train, y_train)

# model evaluation for training set

>> y_train_predict = reg_all.predict(X_train)
>> rmse = (np.sqrt(mean_squared_error(y_train, y_train_predict)))
>> r2 = round(reg_all.score(X_train, y_train),2)

>> print("The model performance for training set")
>> print("--------------------------------------")
>> print('RMSE is {}'.format(rmse))
>> print('R2 score is {}'.format(r2))
>> print("\n")
```

```
X = bos.drop('PRICE', axis = 1)
y = bos['PRICE']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=42)

reg_all = LinearRegression()
reg_all.fit(X_train, y_train)

# model evaluation for training set

y_train_predict = reg_all.predict(X_train)
rmse = (np.sqrt(mean_squared_error(y_train, y_train_predict)))
r2 = round(reg_all.score(X_train, y_train),2)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")
```

```
The model performance for training set
--------------------------------------
RMSE is 4.6520331848801675
R2 score is 0.75
```

```
>> y_pred = reg_all.predict(X_test)
>> rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
>> r2 = round(reg_all.score(X_test, y_test),2)

>> print("The model performance for training set")
>> print("--------------------------------------")
>> print("Root Mean Squared Error: {}".format(rmse))
>> print("R^2: {}".format(r2))
>> print("\n")
```

```
y_pred = reg_all.predict(X_test)
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
r2 = round(reg_all.score(X_test, y_test),2)

print("The model performance for training set")
print("--------------------------------------")
print("Root Mean Squared Error: {}".format(rmse))
print("R^2: {}".format(r2))
print("\n")
```

```
The model performance for training set
--------------------------------------
Root Mean Squared Error: 4.928602182665329
R^2: 0.67
```

```
>> plt.scatter(y_test, y_pred)
>> plt.xlabel("Actual House Prices ($1000)")
>> plt.ylabel("Predicted House Prices: ($1000)")
```

```
>> plt.xticks(range(0, int(max(y_test)),2))
>> plt.yticks(range(0, int(max(y_test)),2))
>> plt.title("Actual Prices vs Predicted prices")
```

```
Text(0.5, 1.0, 'Actual Prices vs Predicted prices')
```

Practical No. 7: Information Extraction

Aim: Write a program for Information extraction.

Code:

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

1. **Part-of-Speech Tagging**

   The part of speech explains how a word is used in a sentence. In a sentence, a word can have different contexts and semantic meanings. The basic natural language processing models like bag-of-words fail to identify these relations between words. Hence, we use part of speech tagging to mark a word to its part of speech tag based on its context in the data. It is also used to extract relationships between words.

   Code:

```
>> nltk.download('punkt')
>> nltk.download('averaged_perceptron_tagger')
>> from nltk.tokenize import word_tokenize

>> from nltk import pos_tag

# convert text into word_tokens with their tags

>> def pos_tagging(text):

>>  word_tokens = word_tokenize(text)

>>  return pos_tag(word_tokens)

>> pos_tagging('You just gave me a scare')

# download the tagset
>> nltk.download('tagsets')
# extract information about the tag
>> nltk.help.upenn_tagset('NN')
```

```
[ ] from nltk.tokenize import word_tokenize
    from nltk import pos_tag

    # convert text into word_tokens with their tags
    def pos_tagging(text):
      word_tokens = word_tokenize(text)
      return pos_tag(word_tokens)

    pos_tagging('You just gave me a scare')

    [('You', 'PRP'),
     ('just', 'RB'),
     ('gave', 'VBD'),
     ('me', 'PRP'),
     ('a', 'DT'),
     ('scare', 'NN')]
```

```
[ ] # download the tagset
    nltk.download('tagsets')

    [nltk_data] Downloading package tagsets to /root/nltk_data...
    [nltk_data]   Package tagsets is already up-to-date!
    True
```

```
    # extract information about the tag
    nltk.help.upenn_tagset('NN')

    NN: noun, common, singular or mass
        common-carrier cabbage knuckle-duster Casino afghan shed thermostat
        investment slide humour falloff slick wind hyena override subhumanity
        machinist ...
```

2. Chunking

Chunking is the process of extracting phrases from unstructured text and more structure to it. It is also known as shallow parsing. It is done on top of Part of Speech tagging. It groups word into "chunks", mainly of noun phrases. Chunking is done using regular expressions.

Code:

```
# Laading Library
>> from nltk.chunk.regexp import tag_pattern2re_pattern

# Chunk Pattern to RegEx Pattern
>> print("Chunk Pattern : ", tag_pattern2re_pattern('<DT>?<NN.*>+'))
>> locs = [('Omnicom', 'IN', 'New York'),
...          ('DDB Needham', 'IN', 'New York'),
...          ('Kaplan Thaler Group', 'IN', 'New York'),
...          ('BBDO South', 'IN', 'Atlanta'),
...          ('Georgia-Pacific', 'IN', 'Atlanta')]
>> query = [e1 for (e1, rel, e2) in locs if e2=='Atlanta']
>> print(query)

>> def ie_preprocess(document):
...     sentences = nltk.sent_tokenize(document)
...     sentences = [nltk.word_tokenize(sent) for sent in sentences]
...     sentences = [nltk.pos_tag(sent) for sent in sentences]

>> sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"),
... ("dog", "NN"), ("barked", "VBD"), ("at", "IN"),  ("the", "DT"), ("c
at", "NN")]
>> grammar = "NP: {<DT>?<JJ>*<NN>}"
>> cp = nltk.RegexpParser(grammar)
```

```
>> result = cp.parse(sentence)
>> print(result)
```

```
locs = [('Omnicom', 'IN', 'New York'),
...         ('DDB Needham', 'IN', 'New York'),
...         ('Kaplan Thaler Group', 'IN', 'New York'),
...         ('BBDO South', 'IN', 'Atlanta'),
...         ('Georgia-Pacific', 'IN', 'Atlanta')]
query = [e1 for (e1, rel, e2) in locs if e2=='Atlanta']
print(query)

def ie_preprocess(document):
...     sentences = nltk.sent_tokenize(document)
...     sentences = [nltk.word_tokenize(sent) for sent in sentences]
...     sentences = [nltk.pos_tag(sent) for sent in sentences]

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)
```

```
['BBDO South', 'Georgia-Pacific']
(S
  (NP the/DT little/JJ yellow/JJ dog/NN)
  barked/VBD
  at/IN
  (NP the/DT cat/NN))
```

3. Chinking

Chinking is a lot like chunking, it is basically a way for you to remove a chunk from a chunk. The chunk that you remove from your chunk is your chink. The code is very similar, you just denote the chink, after the chunk, with }{ instead of the chunk's {}.

Code:

```
>> grammar = r"""
        NP: {<DT|PP\$>?<JJ>*<NN>}
                {<NNP>+}
   """
   >> cp = nltk.RegexpParser(grammar)
   >> sentence = [("Rapunzel", "NNP"), ("let", "VBD"), ("down", "RP"),
         ("her", "PP$"), ("long", "JJ"), ("golden", "JJ"), ("hair", "NN")]
>> print(cp.parse(sentence))
>> nouns = [("money", "NN"), ("market", "NN"), ("fund", "NN")]
>> grammar = "NP: {<NN><NN>}  # Chunk two consecutive nouns"
>> cp = nltk.RegexpParser(grammar)
>> print(cp.parse(nouns))
```

```
grammar = r"""
  NP: {<DT|PP\$>?<JJ>*<NN>}
      {<NNP>+}
"""
cp = nltk.RegexpParser(grammar)
sentence = [("Rapunzel", "NNP"), ("let", "VBD"), ("down", "RP"),
                 ("her", "PP$"), ("long", "JJ"), ("golden", "JJ"), ("hair", "NN")]
print(cp.parse(sentence))
```

```
(S
  (NP Rapunzel/NNP)
  let/VBD
  down/RP
  (NP her/PP$ long/JJ golden/JJ hair/NN))
```

```
nouns = [("money", "NN"), ("market", "NN"), ("fund", "NN")]
grammar = "NP: {<NN><NN>}  # Chunk two consecutive nouns"
cp = nltk.RegexpParser(grammar)
print(cp.parse(nouns))
```

```
(S (NP money/NN market/NN) fund/NN)
```

4. Named Entity Recognition

Named Entity Recognition is used to extract information from unstructured text. It is used to classify entities present in a text into categories like a person, organization, event, places, etc. It gives us detailed knowledge about the text and the relationships between the different entities.

Code:

```
>> nltk.download('words')
>> nltk.download('maxent_ne_chunker')
>> from nltk.tokenize import word_tokenize
>> from nltk import pos_tag, ne_chunk
>> def named_entity_recognition(text):
  # tokenize the text
  >> word_tokens = word_tokenize(text)

  # part of speech tagging of words
  >> word_pos = pos_tag(word_tokens)
  # tree of word entities
  >> print(ne_chunk(word_pos))
>>text = 'Bill works for GeeksforGeeks so he went to Delhi for a meetup
.'
>> named_entity_recognition(text)
```

```
[ ] nltk.download('words')
    nltk.download('maxent_ne_chunker')

    [nltk_data] Downloading package words to /root/nltk_data...
    [nltk_data]   Unzipping corpora/words.zip.
    [nltk_data] Downloading package maxent_ne_chunker to
    [nltk_data]     /root/nltk_data...
    [nltk_data]   Package maxent_ne_chunker is already up-to-date!
    True
```

```
from nltk.tokenize import word_tokenize
from nltk import pos_tag, ne_chunk

def named_entity_recognition(text):
  # tokenize the text
  word_tokens = word_tokenize(text)

  # part of speech tagging of words
  word_pos = pos_tag(word_tokens)

  # tree of word entities
  print(ne_chunk(word_pos))

text = 'Bill works for GeeksforGeeks so he went to Delhi for a meetup.'
named_entity_recognition(text)
```

```
(S
  (PERSON Bill/NNP)
  works/VBZ
  for/IN
  (ORGANIZATION GeeksforGeeks/NNP)
  so/RB
  he/PRP
  went/VBD
  to/TO
```

5. **Relation Extraction**

Once named entities have been identified in a text, we then want to extract the relations that exist between them. As indicated earlier, we will typically be looking for relations between specified types of named entity. One way of approaching this task is to initially look for all triples of the form (X, α, Y), where X and Y are named entities of the required types, and α is the string of words that intervenes between X and Y. We can then use regular expressions to pull out just those instances of α that express the relation that we are looking for. The following example searches for strings that contain the word in. The special regular expression (?!\b.+ing\b) is a negative lookahead assertion that allows us to disregard strings such as success in supervising the transition of, where in is followed by a gerund.

Code:

```
>> nltk.download('ieer')
>> import nltk
>> IN = re.compile(r'.*\bin\b(?!\b.+ing)')
>> for doc in nltk.corpus.ieer.parsed_docs('NYT_19980315'):
>> for rel in nltk.sem.extract_rels('ORG', 'LOC', doc,
                                 corpus='ieer', pattern = IN):
>> print(nltk.sem.rtuple(rel))
>> nltk.download('conll2002')
>> from nltk.corpus import conll2002
>> vnv = """
(
is/V|     # 3rd sing present and
was/V|    # past forms of the verb zijn ('be')
```

```
werd/V|   # and also present
wordt/V   # past of worden ('become)
)
 .*        # followed by anything
van/Prep # followed by van ('of')
"""

>> VAN = re.compile(vnv, re.VERBOSE)
>> for doc in conll2002.chunked_sents('ned.train'):
>>for rel in nltk.sem.extract_rels('PER', 'ORG', doc,

>> corpus='conll2002', pattern=VAN):
>> print(nltk.sem.clause(rel, relsym="VAN"))
```

```
[ ] [ORG: 'Open Text'] ', based in' [LOC: 'Waterloo']
    [ORG: 'WGBH'] 'in' [LOC: 'Boston']
    [ORG: 'Bastille Opera'] 'in' [LOC: 'Paris']
    [ORG: 'Omnicom'] 'in' [LOC: 'New York']
    [ORG: 'DDB Needham'] 'in' [LOC: 'New York']
    [ORG: 'Kaplan Thaler Group'] 'in' [LOC: 'New York']
    [ORG: 'BBDO South'] 'in' [LOC: 'Atlanta']
    [ORG: 'Georgia-Pacific'] 'in' [LOC: 'Atlanta']
```

```
[ ] nltk.download('conll2002')

    [nltk_data] Downloading package conll2002 to /root/nltk_data...
    [nltk_data]   Package conll2002 is already up-to-date!
    True
```

```
from nltk.corpus import conll2002
vnv = """
(
is/V|     # 3rd sing present and
was/V|    # past forms of the verb zijn ('be')
werd/V|   # and also present
wordt/V   # past of worden ('become)
)
 .*        # followed by anything
van/Prep # followed by van ('of')
"""

VAN = re.compile(vnv, re.VERBOSE)
for doc in conll2002.chunked_sents('ned.train'):
 for rel in nltk.sem.extract_rels('PER', 'ORG', doc,
                                  corpus='conll2002', pattern=VAN):
  print(nltk.sem.clause(rel, relsym="VAN"))
```

```
VAN("cornet_d'elzius", 'buitenlandse_handel')
VAN('johan_rottiers', 'kardinaal_van_roey_instituut')
VAN('annie_lennox', 'eurythmics')
```

Practical No. 8: Classification

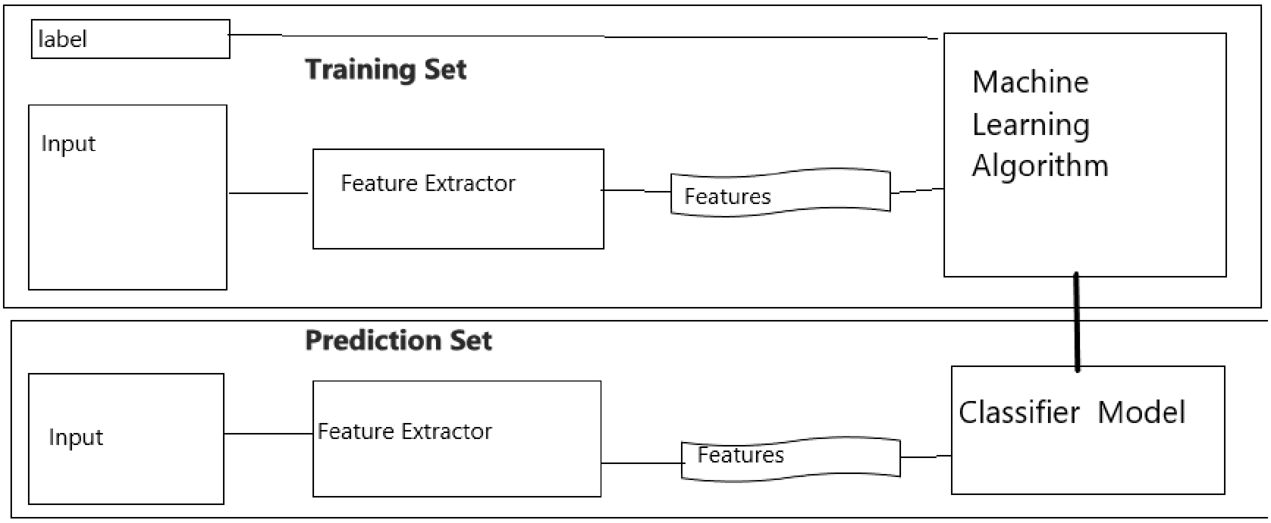
Aim: Write a program for classification.

Classification is the task of choosing the correct class label for a given input. In basic classification tasks, each input is considered in isolation from all other inputs, and the set of labels is defined in advance. Some examples of classification tasks are:

1. Deciding whether an email is spam or not.
2. Deciding what the topic of a news article is, from a fixed list of topic areas such as “sports, ” “technology, ” and “politics.”
3. Deciding whether a given occurrence of the word bank is used to refer to a river bank, a financial institution, the act of tilting to the side, or the act of depositing something in a financial institution.

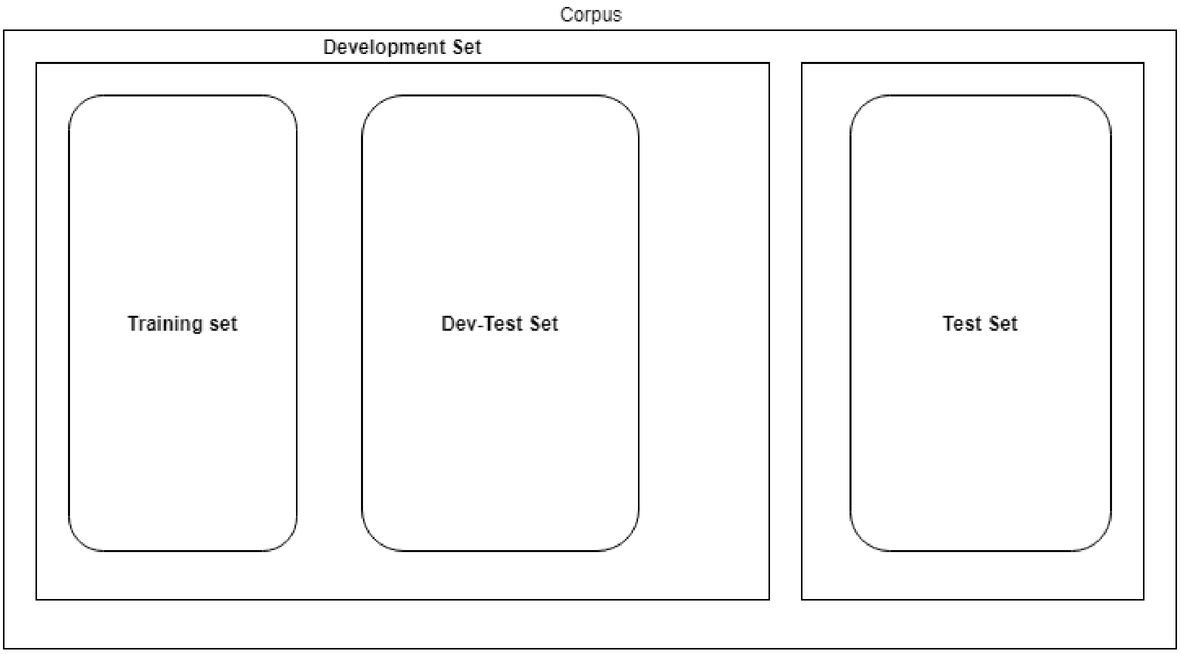
The basic classification task has a number of interesting variants. For example, in multi-class classification, each instance may be assigned multiple labels; in open-class classification, the set of labels is not defined in advance; and in sequence classification, a list of inputs are jointly classified.

A **classifier** is called supervised if it is built based on training corpora containing the correct label for each input. The **framework** used by supervised classification is shown in figure.



The training set is used to train the model, and the dev-test set is used to perform error analysis. The test set serves in our final evaluation of the system. For reasons discussed below, it is important that we employ a separate dev-test set for error analysis, rather than just using the test set.

The division of the corpus data into different subsets is shown in following Figure:



Code :

4. Supervised Classification - Gender Identification

```
>> def gender_features(word):
>> return {'last_letter': word[-1]}
>> gender_features('Shrek')
>> import nltk
>> nltk.download('names')
>> from nltk.corpus import names
>>labeled_names = ([(name, 'male') for name in names.words('male.txt')]
 + [(name, 'female') for name in names.words('female.txt')])
>> import random
>> random.shuffle(labeled_names)
>>featuresets = [(gender_features(n), gender) for (n, gender) in labele
d_names]
>> train_set, test_set = featuresets[500:], featuresets[:500]
>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>> classifier.classify(gender_features('Neo'))
>> classifier.classify(gender_features('Trinity'))
>> print(nltk.classify.accuracy(classifier, test_set))
>> classifier.show_most_informative_features(5)
```

```
classifier.classify(gender_features('Neo'))
```

```
'male'
```

```
[4] classifier.classify(gender_features('Trinity'))
```

```
'female'
```

```
[5] print(nltk.classify.accuracy(classifier, test_set))
```

```
0.774
```

```
classifier.show_most_informative_features(5)
```

```
Most Informative Features
             last_letter = 'k'              male : female =     45.4 : 1.0
             last_letter = 'a'            female : male   =     36.9 : 1.0
             last_letter = 'f'              male : female =     16.0 : 1.0
             last_letter = 'p'              male : female =     12.6 : 1.0
             last_letter = 'd'              male : female =      9.9 : 1.0
```

5. Choosing right features

```
>> from nltk.classify import apply_features
>> train_set = apply_features(gender_features, labeled_names[500:])
>> test_set = apply_features(gender_features, labeled_names[:500])

>> def gender_features2(name):
    features = {}
    features["first_letter"] = name[0].lower()
    features["last_letter"] = name[-1].lower()
    for letter in 'abcdefghijklmnopqrstuvwxyz':
        features["count({})".format(letter)] = name.lower().count(lette
r)
        features["has({})".format(letter)] = (letter in name.lower())
    return features
>> gender_features2('John')
```

```
from nltk.classify import apply_features
train_set = apply_features(gender_features, labeled_names[500:])
test_set = apply_features(gender_features, labeled_names[:500])
```

```
[9] def gender_features2(name):
        features = {}
        features["first_letter"] = name[0].lower()
        features["last_letter"] = name[-1].lower()
        for letter in 'abcdefghijklmnopqrstuvwxyz':
            features["count({})".format(letter)] = name.lower().count(letter)
            features["has({})".format(letter)] = (letter in name.lower())
        return features
```

```
gender_features2('John')
```

```
{'count(a)': 0,
 'count(b)': 0,
 'count(c)': 0,
 'count(d)': 0,
 'count(e)': 0,
 'count(f)': 0,
 'count(g)': 0,
 'count(h)': 1,
 'count(i)': 0,
 'count(j)': 1,
 'count(k)': 0,
 'count(l)': 0,
 'count(m)': 0,
 'count(n)': 1,
 'count(o)': 1,
 'count(p)': 0,
 'count(q)': 0,
 'count(r)': 0,
 'count(s)': 0,
```

```
>>featuresets = [(gender_features2(n), gender) for (n, gender) in labeled_names]
>> train_set, test_set = featuresets[500:], featuresets[:500]
>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>> print(nltk.classify.accuracy(classifier, test_set))
>> train_names = labeled_names[1500:]
>> devtest_names = labeled_names[500:1500]
>> test_names = labeled_names[:500]
>>train_set = [(gender_features(n), gender) for (n, gender) in train_names]
>>devtest_set = [(gender_features(n), gender) for (n, gender) in devtest_names]
>>test_set = [(gender_features(n), gender) for (n, gender) in test_names]
>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>> print(nltk.classify.accuracy(classifier, devtest_set))
```

```
featuresets = [(gender_features2(n), gender) for (n, gender) in labeled_names]
train_set, test_set = featuresets[500:], featuresets[:500]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, test_set))
```

```
0.78
```

```
[12] train_names = labeled_names[1500:]
     devtest_names = labeled_names[500:1500]
     test_names = labeled_names[:500]
```

```
[13] train_set = [(gender_features(n), gender) for (n, gender) in train_names]
     devtest_set = [(gender_features(n), gender) for (n, gender) in devtest_names]
     test_set = [(gender_features(n), gender) for (n, gender) in test_names]
     classifier = nltk.NaiveBayesClassifier.train(train_set)
     print(nltk.classify.accuracy(classifier, devtest_set))
```

```
0.765
```

```
errors = []
for (name, tag) in devtest_names:
 guess = classifier.classify(gender_features(name))
if guess != tag:
 errors.append( (tag, guess, name) )
```

```
[15] for (tag, guess, name) in sorted(errors):
         print('correct={:<8} guess={:<8s} name={:<30}'.format(tag, guess, name))
```

```
[16] def gender_features(word):
      return {'suffix1': word[-1:],
     'suffix2': word[-2:]}
```

```
train_set = [(gender_features(n), gender) for (n, gender) in train_names]
devtest_set = [(gender_features(n), gender) for (n, gender) in devtest_names]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, devtest_set))
```

```
0.779
```

6. Document classification

```
>> import nltk
>> nltk.download('movie_reviews')
>> from nltk.corpus import movie_reviews
>> documents = [(list(movie_reviews.words(fileid)), category)
>> for category in movie_reviews.categories()
>> for fileid in movie_reviews.fileids(category)]
>> random.shuffle(documents)
>> all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
>> word_features = list(all_words)[:2000]

>> def document_features(document):
```

```
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = (word in document_words
)
    return features
>> print(document_features(movie_reviews.words('pos/cv957_8737.txt')))
>> featuresets = [(document_features(d), c) for (d,c) in documents]
>> train_set, test_set = featuresets[100:], featuresets[:100]
>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>> print(nltk.classify.accuracy(classifier, test_set))
>> classifier.show_most_informative_features(5)
```

```
[22] featuresets = [(document_features(d), c) for (d,c) in documents]
     train_set, test_set = featuresets[100:], featuresets[:100]
     classifier = nltk.NaiveBayesClassifier.train(train_set)
```

```
[23] print(nltk.classify.accuracy(classifier, test_set))

     0.8
```

```
classifier.show_most_informative_features(5)

Most Informative Features
 contains(unimaginative) = True              neg : pos    =      8.4 : 1.0
       contains(martian) = True              neg : pos    =      7.7 : 1.0
        contains(suvari) = True              neg : pos    =      7.0 : 1.0
        contains(shoddy) = True              neg : pos    =      7.0 : 1.0
          contains(mena) = True              neg : pos    =      7.0 : 1.0
```

7. Sentence Segmentation

```
>> nltk.download('treebank')

>> import nltk
>> nltk.download('punkt')
>> import nltk
>> sents = nltk.corpus.treebank_raw.sents()
>> tokens = []
>> boundaries = set()
>> offset = 0
>> for sent in sents:
>> tokens.extend(sent)
>> offset += len(sent)
>> boundaries.add(offset-1)
>> def punct_features(tokens, i):
>> return {'next-word-capitalized': tokens[i+1][0].isupper(),
'prev-word': tokens[i-1].lower(),
'punct': tokens[i],
'prev-word-is-one-char': len(tokens[i-1]) == 1}
>> featuresets = [(punct_features(tokens, i), (i in boundaries))
```

```
>> for i in range(1, len(tokens)-1)
>> if tokens[i] in '.?!']
>> size = int(len(featuresets) * 0.1)
>> train_set, test_set = featuresets[size:], featuresets[:size]
>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>> nltk.classify.accuracy(classifier, test_set)
```

```
[29] offset = 0
```

```
[30] for sent in sents:
       tokens.extend(sent)
     offset += len(sent)
     boundaries.add(offset-1)
```

```
[31] def punct_features(tokens, i):
      return {'next-word-capitalized': tokens[i+1][0].isupper(),
     'prev-word': tokens[i-1].lower(),
     'punct': tokens[i],
     'prev-word-is-one-char': len(tokens[i-1]) == 1}
```

```
featuresets = [(punct_features(tokens, i), (i in boundaries))
 for i in range(1, len(tokens)-1)
if tokens[i] in '.?!']
```

```
size = int(len(featuresets) * 0.1)
train_set, test_set = featuresets[size:], featuresets[:size]
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, test_set)
```

```
1.0
```

8. Naive Bayes Classifier

Naive Bayes classification is a fast and simple to understand classification method. Its speed is due to some simplifications we make about the underlying probability distributions, namely, the assumption about the independence of features. Yet, it can be quite powerful, especially when there are enough features in the data.
Suppose we have for each label L a probability distribution. This distribution gives probability for each possible combination of features (a feature vector):

$$P(features|L).$$

The main idea in Bayesian classification is to reverse the direction of dependence: we want to predict the label based on the features:

$$P(L|features)$$

This is possible by the Bayes theorem:

$$P(L|features)=P(features|L)P(L)P(features).$$

Let's assume we have to labels L1 and L2, and their associated distributions: P(features|L1) and P(features|L2). If we have a data point with "features", whose label we don't know, we can try to predict it using the ratio of posterior probabilities:

$$P(L1|features)P(L2|features)=P(features|L1)P(L1)P(features|L2)P(L2).$$

If the ratio is greater than one, we label our data point with label L1, and if not, we give it label L2. The prior probabilities P(L1) and P(L2) of labels can be easily found out from the input data, as for each data point we also have its label. Same goes for the probabilities of features conditioned on the label.

Code:

>We first demonstrate naive Bayes classification using Gaussian distributions.

```
1]
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
```

```
2]
from sklearn.datasets import make_blobs
X,y = make_blobs(100, 2, centers=2, random_state=2, cluster_std=1.5)
colors=np.array(["red", "blue"])
plt.scatter(X[:, 0], X[:, 1], c=colors[y], s=50)
for label, c in enumerate(colors):
    plt.scatter([], [], c=c, label=str(label))
plt.legend();
#plt.colorbar();
```

3]

```
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
model = GaussianNB()
#model = MultinomialNB()
model.fit(X, y);
```

>Naive Bayes algorithm fitted two 2-dimensional Gaussian distribution to the data. The means and the variances define these distributions completely.

1]

```
print("Means:", model.theta_)
print("Standard deviations:", model.sigma_)
```

2]

```
def plot_ellipse(ax, mu, sigma, color="k", label=None):
    """
    Based on
    http://stackoverflow.com/questions/17952171/not-sure-how-to-fit-
data-with-a-gaussian-python.
    """
    from matplotlib.patches import Ellipse
    # Compute eigenvalues and associated eigenvectors
    vals, vecs = np.linalg.eigh(sigma)
```

```
    # Compute "tilt" of ellipse using first eigenvector
    x, y = vecs[:, 0]
    theta = np.degrees(np.arctan2(y, x))

    # Eigenvalues give length of ellipse along each eigenvector
    w, h = 2 * np.sqrt(vals)

    ax.tick_params(axis='both', which='major', labelsize=20)
    ellipse = Ellipse(mu, w, h, theta, color=color, label=label)  # col
or="k")
    ellipse.set_clip_box(ax.bbox)
    ellipse.set_alpha(0.2)
    ax.add_artist(ellipse)
    return ellipse
```

3]

```
plt.figure()
plt.xlim(-5, 5)
plt.ylim(-15, 5)
plot_ellipse(plt.gca(), model.theta_[0], np.identity(2)*model.sigma_[0]
, color="red")
plot_ellipse(plt.gca(), model.theta_[1], np.identity(2)*model.sigma_[1]
, color="blue");
```

4]

```
from sklearn.metrics import accuracy_score
y_fitted = model.predict(X)
acc=accuracy_score(y,y_fitted)
print("Accuracy score is", acc)
```

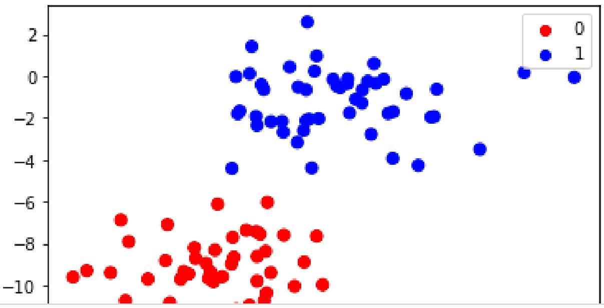
Naive Bayes.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
```

```
[2] from sklearn.datasets import make_blobs
    X,y = make_blobs(100, 2, centers=2, random_state=2, cluster_std=1.5)
    colors=np.array(["red", "blue"])
    plt.scatter(X[:, 0], X[:, 1], c=colors[y], s=50)
    for label, c in enumerate(colors):
        plt.scatter([], [], c=c, label=str(label))
    plt.legend();
    #plt.colorbar();
```

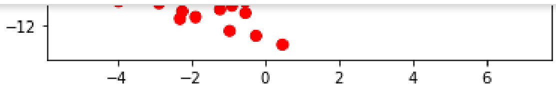


Naive Bayes.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[2]




```
[3] from sklearn.naive_bayes import GaussianNB
    from sklearn.naive_bayes import MultinomialNB
    model = GaussianNB()
    #model = MultinomialNB()
    model.fit(X, y);
```

```
[4] print("Means:", model.theta_)
    print("Standard deviations:", model.sigma_)
```

```
Means: [[-1.64939095 -9.36891451]
 [ 1.29327924 -1.24101221]]
Standard deviations: [[2.06097005 2.47716872]
 [3.33164807 2.22401384]]
```

```
[5] def plot_ellipse(ax, mu, sigma, color="k", label=None):
        """
        Based on
        http://stackoverflow.com/questions/17952171/not-sure-how-to-fit-data-with-a-gaussian-p
        """
        from matplotlib.patches import Ellipse
        # Compute eigenvalues and associated eigenvectors
```

Naive Bayes.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[5]     # Compute eigenvalues and associated eigenvectors
        vals, vecs = np.linalg.eigh(sigma)

        # Compute "tilt" of ellipse using first eigenvector
        x, y = vecs[:, 0]
        theta = np.degrees(np.arctan2(y, x))

        # Eigenvalues give length of ellipse along each eigenvector
        w, h = 2 * np.sqrt(vals)

        ax.tick_params(axis='both', which='major', labelsize=20)
        ellipse = Ellipse(mu, w, h, theta, color=color, label=label)  # color="k")
        ellipse.set_clip_box(ax.bbox)
        ellipse.set_alpha(0.2)
        ax.add_artist(ellipse)
        return ellipse
```

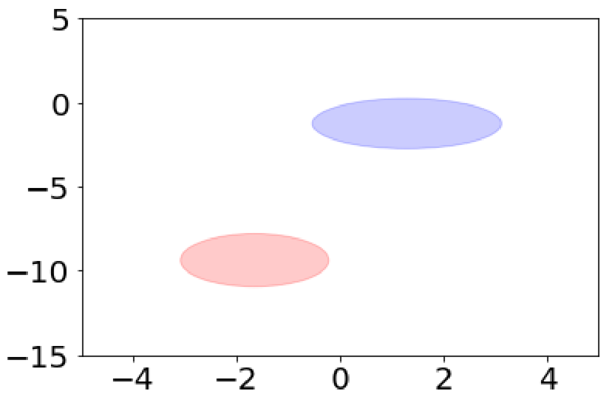
```
[6] plt.figure()
    plt.xlim(-5, 5)
    plt.ylim(-15, 5)
    plot_ellipse(plt.gca(), model.theta_[0], np.identity(2)*model.sigma_[0], color="red")
    plot_ellipse(plt.gca(), model.theta_[1], np.identity(2)*model.sigma_[1], color="blue");
```

Naive Bayes.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[6] plot_ellipse(plt.gca(), model.theta_[1], np.identity(2)*model.sigma_[1], color="blue");
```




```
from sklearn.metrics import accuracy_score
y_fitted = model.predict(X)
acc=accuracy_score(y,y_fitted)
print("Accuracy score is", acc)
```

```
Accuracy score is 1.0
```

A. Using Wordnet Finding Synonym and Antonym

WordNet's structure makes it a useful tool for computational linguistics and natural language processing.

WordNet superficially resembles a thesaurus, in that it groups words together based on their meanings. However, there are some important distinctions.
First, WordNet interlinks not just word forms—strings of letters—but specific senses of words. As a result, words that are found in close proximity to one another in the network are semantically disambiguated.
Second, WordNet labels the semantic relations among words, whereas the groupings of words in a thesaurus does not follow any explicit pattern other than meaning similarity.

Code:

```
>> import nltk
>> nltk.download('wordnet')
# First, you're going to need to import wordnet:
>> from nltk.corpus import wordnet
# Then, we're going to use the term "program" to find synsets like so:
>> syns = wordnet.synsets("program")
# An example of a synset:
>> print(syns[0].name())
# Just the word:
>> print(syns[0].lemmas()[0].name())
# Definition of that first synset:
>> print(syns[0].definition())
# Examples of the word in use in sentences:
>> print(syns[0].examples())
>> import nltk
>> from nltk.corpus import wordnet
>> synonyms = []
>> antonyms = []
>> for syn in wordnet.synsets("good"):
   for l in syn.lemmas():
     synonyms.append(l.name())
     if l.antonyms():
       antonyms.append(l.antonyms()[0].name())

>> print(set(synonyms))
>> print(set(antonyms))
>> import nltk
>> from nltk.corpus import wordnet
>> synonyms = []
>> antonyms = []

>> for syn in wordnet.synsets("good"):
   for l in syn.lemmas():
     synonyms.append(l.name())
     if l.antonyms():
       antonyms.append(l.antonyms()[0].name())

>> print(set(synonyms))
>> print(set(antonyms))
```

```
>> import nltk
>> from nltk.corpus import wordnet
# Let's compare the noun of "ship" and "boat:"

>> w1 = wordnet.synset('run.v.01') # v here denotes the tag verb
.01')
>> print(w1.wup_similarity(w2))
>> w1 = wordnet.synset('ship.n.01')
>> w2 = wordnet.synset('boat.n.01') # n denotes noun
>> print(w1.wup_similarity(w2))
```

```
# First, you're going to need to import wordnet:
from nltk.corpus import wordnet

# Then, we're going to use the term "program" to find synsets like so:
syns = wordnet.synsets("program")

# An example of a synset:
print(syns[0].name())

# Just the word:
print(syns[0].lemmas()[0].name())

# Definition of that first synset:
print(syns[0].definition())

# Examples of the word in use in sentences:
print(syns[0].examples())
```

```
plan.n.01
plan
a series of steps to be carried out or goals to be accomplished
['they drew up a six-step plan', 'they discussed plans for a new bond issue']
```

```python
import nltk
from nltk.corpus import wordnet
synonyms = []
antonyms = []

for syn in wordnet.synsets("good"):
  for l in syn.lemmas():
    synonyms.append(l.name())
    if l.antonyms():
      antonyms.append(l.antonyms()[0].name())

print(set(synonyms))
print(set(antonyms))
```

```
{'trade_good', 'ripe', 'adept', 'good', 'upright', 'safe', 'e
{'badness', 'evilness', 'evil', 'bad', 'ill'}
```

```python
import nltk
from nltk.corpus import wordnet
# Let's compare the noun of "ship" and "boat:"

w1 = wordnet.synset('run.v.01') # v here denotes the tag verb
w2 = wordnet.synset('sprint.v.01')
print(w1.wup_similarity(w2))
```

```python
print(set(synonyms))
print(set(antonyms))
```

```
{'trade_good', 'ripe', 'adept', 'good', 'upright', 'safe', 'expert', 'effective
{'badness', 'evilness', 'evil', 'bad', 'ill'}
```

```python
import nltk
from nltk.corpus import wordnet
# Let's compare the noun of "ship" and "boat:"

w1 = wordnet.synset('run.v.01') # v here denotes the tag verb
w2 = wordnet.synset('sprint.v.01')
print(w1.wup_similarity(w2))
```

```
0.8571428571428571
```

```python
w1 = wordnet.synset('ship.n.01')
w2 = wordnet.synset('boat.n.01') # n denotes noun
print(w1.wup_similarity(w2))
```

```
0.9090909090909091
```

B. Word Sense Disambiguation - Leak Algorithm

To use Python code to remove word ambiguity using the Lesk algorithm.

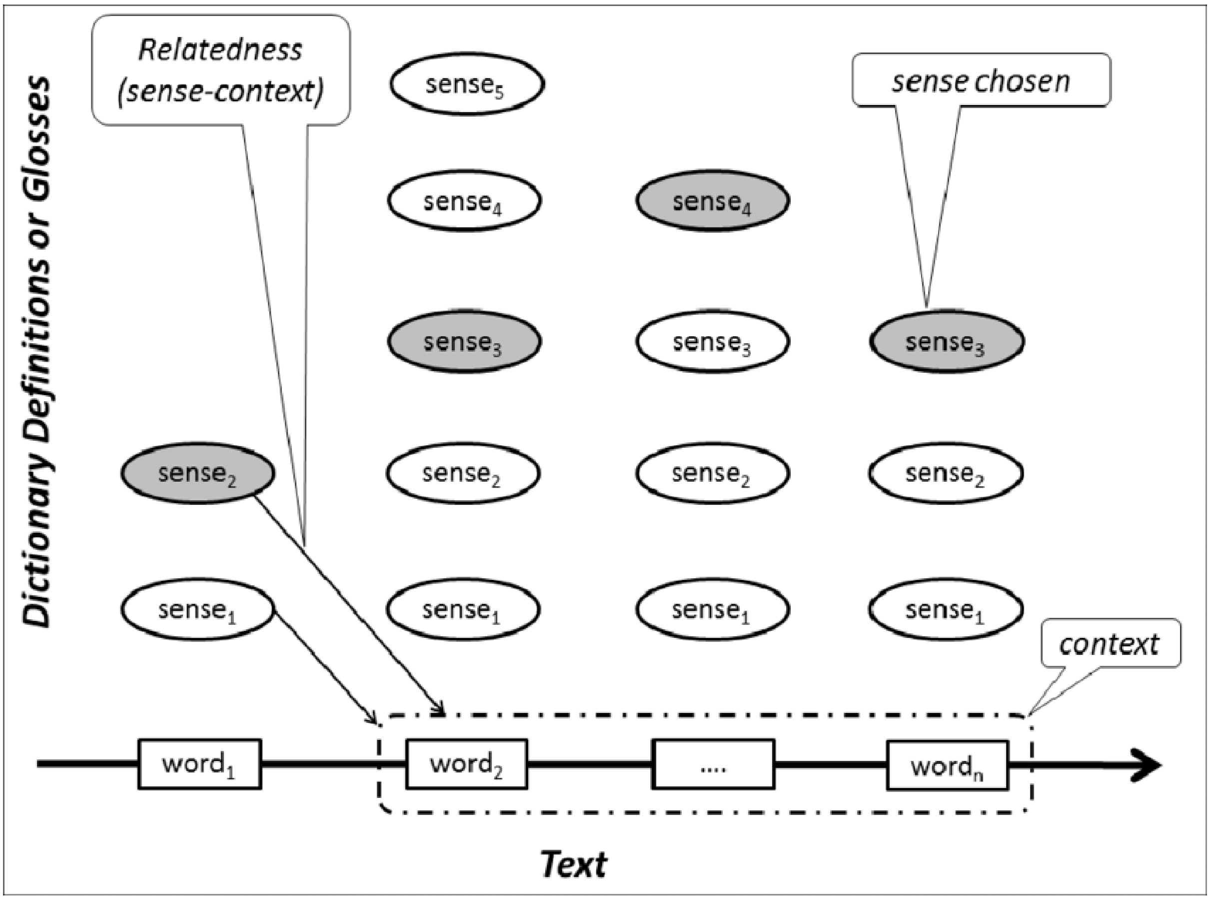
For example, in the sentences below, the word "bank" has different meanings based on the context of the sentence.

Text1 = 'I went to the bank to deposit my money'
Text2 = 'The river bank was full of dead fishes'

The Lesk algorithm is the seminal dictionary-based method.

It is based on the hypothesis that words used together in text are related to each other and that the relation can be observed in the definitions of the words and their senses. Two (or more) words are disambiguated by finding the pair of dictionary senses with the greatest word overlap in their dictionary definitions. It searches for the shortest path between two words: the second word is iteratively searched among the definitions of every semantic variant of the first word, then among the definitions of every semantic variant of each word in the previous definitions and so on.
Finally, the first word is disambiguated by selecting the semantic variant which minimizes the distance from the first to the second word."

Basically, the context is chosen from meaning of the nearest words. Following is the simplified pictorial representation of the same...



Let's see the code to implement the Lesk algorithm in Python.

First install the library pywsd - python implementation of Word Sense Disambiguation (WSD)

1]

```
pip install pywsd==1.1.1
```

2]

```
import nltk
nltk.download('popular')
```

3]

```
import wordnet as wn
from pywsd.lesk import simple_lesk
sentences = ['The workers at the plant were overworked',
'The plant was no longer bearing flowers',
'The workers at the industrial plant were overworked']
# calling the lesk function and printing results for both the sentences

print ("Context-1:", sentences[0])
answer = simple_lesk(sentences[0],'plant')
print ("Sense:", answer)
print ("Definition : ", answer.definition())
```

Output:

```
Context-1: The workers at the plant were overworked
Sense: Synset('plant.v.06')
Definition :  put firmly in the mind
```

4]

```
# calling the lesk function and printing results
print ("Context-1:", sentences[1])
answer = simple_lesk(sentences[1],'plant')
print ("Sense:", answer)
print ("Definition : ", answer.definition())
```

output:

```
Context-1: The plant was no longer bearing flowers
Sense: Synset('plant.v.01')
Definition :  put or set (seeds, seedlings, or plants) into the ground
```

5]

```
print ("Context-3:", sentences[2])
answer = simple_lesk(sentences[2],'plant')
print ("Sense:", answer)
print ("Definition : ", answer.definition())
```

output:

```
Context-3: The workers at the industrial plant were overworked
Sense: Synset('plant.n.01')
Definition :  buildings for carrying on industrial labor
```

Lesk Algorithm.ipynb

File Edit View Insert Runtime Tools Help All changes saved

\+ Code + Text

```
pip install pywsd==1.1.1
```

```
Collecting pywsd==1.1.1
  Downloading https://files.pythonhosted.org/packages/d9/16/3b7c1348c010a6b156dd1ab1791038b34b7b876e787
Requirement already satisfied: nltk in /usr/local/lib/python3.7/dist-packages (from pywsd==1.1.1) (3.6.
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from pywsd==1.1.1) (1.1
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from nltk->pywsd==1.1.
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from nltk->pywsd==1.1.1)
Requirement already satisfied: regex in /usr/local/lib/python3.7/dist-packages (from nltk->pywsd==1.1.1
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from nltk->pywsd==1.1.1
Building wheels for collected packages: pywsd
  Building wheel for pywsd (setup.py) ... done
  Created wheel for pywsd: filename=pywsd-1.1.1-cp37-none-any.whl size=12805 sha256=109d58776d8050509b4
  Stored in directory: /root/.cache/pip/wheels/b8/11/47/cba87016528b404a642fa607d0811e9c577fd90eee8dd05
Successfully built pywsd
Installing collected packages: pywsd
  Found existing installation: pywsd 1.0
    Uninstalling pywsd-1.0:
      Successfully uninstalled pywsd-1.0
Successfully installed pywsd-1.1.1
```

```
[41] import nltk
     nltk.download('popular')
```

```
[nltk_data] Downloading collection 'popular'
```

Lesk Algorithm.ipynb

File Edit View Insert Runtime Tools Help All changes saved

\+ Code + Text

```
[41] import nltk
     nltk.download('popular')
```

```
[nltk_data] Downloading collection 'popular'
[nltk_data]    |
[nltk_data]    | Downloading package cmudict to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cmudict.zip.
[nltk_data]    | Downloading package gazetteers to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/gazetteers.zip.
[nltk_data]    | Downloading package genesis to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/genesis.zip.
[nltk_data]    | Downloading package gutenberg to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/gutenberg.zip.
[nltk_data]    | Downloading package inaugural to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/inaugural.zip.
[nltk_data]    | Downloading package movie_reviews to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/movie_reviews.zip.
[nltk_data]    | Downloading package names to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/names.zip.
[nltk_data]    | Downloading package shakespeare to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/shakespeare.zip.
[nltk_data]    | Downloading package stopwords to /root/nltk_data...
[nltk_data]    |   Package stopwords is already up-to-date!
[nltk_data]    | Downloading package treebank to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/treebank.zip.
[nltk_data]    | Downloading package twitter_samples to
```

Lesk Algorithm.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[47]

```
import wordnet as wn
from pywsd.lesk import simple_lesk
sentences = ['The workers at the plant were overworked',
'The plant was no longer bearing flowers',
'The workers at the industrial plant were overworked']
# calling the lesk function and printing results for both the sentences
print ("Context-1:", sentences[0])
answer = simple_lesk(sentences[0],'plant')
print ("Sense:", answer)
print ("Definition : ", answer.definition())
```

```
Context-1: The workers at the plant were overworked
Sense: Synset('plant.v.06')
Definition :  put firmly in the mind
```

```
# calling the lesk function and printing results
print ("Context-1:", sentences[1])
answer = simple_lesk(sentences[1],'plant')
print ("Sense:", answer)
print ("Definition : ", answer.definition())
```

```
Context-1: The plant was no longer bearing flowers
Sense: Synset('plant.v.01')
Definition :  put or set (seeds, seedlings, or plants) into the ground
```

Lesk Algorithm.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
print ("Context-1:", sentences[1])
answer = simple_lesk(sentences[1],'plant')
print ("Sense:", answer)
print ("Definition : ", answer.definition())
```

```
Context-1: The plant was no longer bearing flowers
Sense: Synset('plant.v.01')
Definition :  put or set (seeds, seedlings, or plants) into the ground
```


[49]

```
print ("Context-3:", sentences[2])
answer = simple_lesk(sentences[2],'plant')
print ("Sense:", answer)
print ("Definition : ", answer.definition())
```

```
Context-3: The workers at the industrial plant were overworked
Sense: Synset('plant.n.01')
Definition :  buildings for carrying on industrial labor
```

C. Word2Vec

Word2vec is a technique for natural language processing. The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. ... As the name implies, word2vec represents each distinct word with a particular list of numbers called a vector.

Code:

```
import nltk
nltk.download('brown')
import nltk
import nltk
nltk.download('punkt')
nltk.download('movie_reviews')
import nltk
nltk.download('treebank')
from gensim.models import Word2Vec
from nltk.corpus import brown, movie_reviews, treebank
b = Word2Vec(brown.sents())
mr = Word2Vec(movie_reviews.sents())
t = Word2Vec(treebank.sents())
b.most_similar('money', topn=5)
[('pay', 0.6832243204116821), ('ready', 0.6152011156082153), ('try', 0.
5845392942428589), ('care', 0.5826011896133423), ('move', 0.57521712779
99878)]
mr.most_similar('money', topn=5)
[('unstoppable', 0.6900672316551208), ('pain', 0.6289106607437134), ('o
btain', 0.62665855884552), ('jail', 0.6140228509902954), ('patients', 0
.6089504957199097)]
t.most_similar('money', topn=5)
[('short-term', 0.9459682106971741), ('-LCB-
', 0.9449775218963623), ('rights', 0.9442864656448364), ('interested',
0.9430986642837524), ('national', 0.9396077990531921)]
b.most_similar('great', topn=5)
[('new', 0.6999611854553223), ('experience', 0.6718623042106628), ('soc
ial', 0.6702290177345276), ('group', 0.6684836149215698), ('life', 0.66
67487025260925)]
mr.most_similar('great', topn=5)
[('wonderful', 0.7548679113388062), ('good', 0.6538234949111938), ('str
ong', 0.6523671746253967), ('phenomenal', 0.6296845078468323), ('fine',
 0.5932096242904663)]
t.most_similar('great', topn=5)
[('won', 0.9452997446060181), ('set', 0.9445616006851196), ('target', 0
.9342271089553833), ('received', 0.9333916306495667), ('long', 0.922469
1390991211)]
b.most_similar('company', topn=5)
[('industry', 0.6164317727088928), ('technical', 0.6059585809707642), (
'orthodontist', 0.5982754826545715), ('foamed', 0.5929019451141357), ('
trail', 0.5763031840324402)]
mr.most_similar('company', topn=5)
[('colony', 0.6689200401306152), ('temple', 0.6546304225921631), ('arri
val', 0.6497283577919006), ('army', 0.6339291334152222), ('planet', 0.6
184555292129517)]
t.most_similar('company', topn=5)
```

```
[('panel', 0.7949466705322266), ('Herald', 0.7674347162246704), ('Analysts', 0.7463694214820862), ('amendment', 0.7282689809799194), ('Treasury', 0.719698429107666)]
```

```
b.most_similar('money', topn=5)
[('pay', 0.6832243204116821), ('ready', 0.6152011156082153), ('try', 0.5845392942428589), ('care', 0.5826011896133423), ('move', 0.5752171277999878)]
mr.most_similar('money', topn=5)
[('unstoppable', 0.6900672316551208), ('pain', 0.6289106607437134), ('obtain', 0.62665855884552), ('jail', 0.6140228509902954), ('patients', 0.6089504957199097)]
t.most_similar('money', topn=5)
[('short-term', 0.9459682106971741), ('-LCB-', 0.9449775218963623), ('rights', 0.9442864656448364), ('interested', 0.9430986642837524), ('national', 0.9396077990531921)]
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWarning: Call to deprecated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_simil
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DeprecationWarning: Call to deprecated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_simil
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: DeprecationWarning: Call to deprecated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_simil
  """
[('short-term', 0.9459682106971741),
 ('-LCB-', 0.9449775218963623),
 ('rights', 0.9442864656448364),
 ('interested', 0.9430986642837524),
 ('national', 0.9396077990531921)]
```

```
b.most_similar('great', topn=5)
[('new', 0.6999611854553223), ('experience', 0.6718623042106628), ('social', 0.6702290177345276), ('group', 0.6684836149215698), ('life', 0.6667487025260925)]
mr.most_similar('great', topn=5)
[('wonderful', 0.7548679113388062), ('good', 0.6538234949111938), ('strong', 0.6523671746253967), ('phenomenal', 0.6296845078468323), ('fine', 0.5932096242904663)]
t.most_similar('great', topn=5)
[('won', 0.9452997446060181), ('set', 0.9445616006851196), ('target', 0.9342271089553833), ('received', 0.9333916306495667), ('long', 0.9224691390991211)]
```


✓ 0s completed at 2:41 AM


```
b.most_similar('great', topn=5)
[('new', 0.6999611854553223), ('experience', 0.6718623042106628), ('social', 0.6702290177345276), ('group', 0.6684836149215698), ('life', 0.6667487025260925)]
mr.most_similar('great', topn=5)
[('wonderful', 0.7548679113388062), ('good', 0.6538234949111938), ('strong', 0.6523671746253967), ('phenomenal', 0.6296845078468323), ('fine', 0.5932096242904663)]
t.most_similar('great', topn=5)
[('won', 0.9452997446060181), ('set', 0.9445616006851196), ('target', 0.9342271089553833), ('received', 0.9333916306495667), ('long', 0.9224691390991211)]
```



```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWarning: Call to deprecated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_similar() instead).
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DeprecationWarning: Call to deprecated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_similar() instead).
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: DeprecationWarning: Call to deprecated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_similar() instead).
  """
[('won', 0.9452997446060181),
 ('set', 0.9445616006851196),
 ('target', 0.9342271089553833),
 ('received', 0.9333916306495667),
 ('long', 0.9224691390991211)]
```

```
b.most_similar('company', topn=5)
[('industry', 0.6164317727088928), ('technical', 0.6059585809707642), ('orthodontist', 0.5982754826545715), ('foamed', 0.5929019451141357), ('trail', 0.5763031840324402)]
mr.most_similar('company', topn=5)
[('colony', 0.6689200401306152), ('temple', 0.6546304225921631), ('arrival', 0.6497283577919006), ('army', 0.6339291334152222), ('planet', 0.6184555292129517)]
t.most_similar('company', topn=5)
[('panel', 0.7949466705322266), ('Herald', 0.7674347162246704), ('Analysts', 0.7463694214820862), ('amendment', 0.7282689809799194), ('Treasury', 0.719698429107666)]
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWarning: Call to deprecated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_similar() instead).
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DeprecationWarning: Call to deprecated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_similar() instead).
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: DeprecationWarning: Call to deprecated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_similar() instead).
  """
[('panel', 0.7949466705322266),
 ('Herald', 0.7674347162246704),
 ('Analysts', 0.7463694214820862),
 ('amendment', 0.7282689809799194),
 ('Treasury', 0.719698429107666)]
```


✓ 0s completed at 2:41 AM