

Word Sense Disambiguation



Submitted To -	Prof. Balaji Vijaykumar
Submitted By-	Md. Imran Akhtar (4NI23IS109) Neha Sharma (4NI23CI061) Deepankar Gupta (4NI23CI033)

DATE: 16th April 2025

WSD in Computational Linguistics

1. Introduction

In the present age of information technology (IT), the whole world is sharing information using the internet. This information is available in natural language. As naturally understood, all-natural languages have an intrinsic feature called ambiguity. Ambiguity refers to the situation where a word can have multiple meanings. Ambiguity in natural language poses a significant obstacle in Natural Language Processing (NLP). While the human mind can rely on cognition and world knowledge to disambiguate word senses, machines lack the ability to employ cognition and world knowledge, leading to semantic errors and erroneous interpretations in their output. Therefore, the WSD process is employed to alleviate ambiguity in sentences.

WSD represents highly regarded formidable challenges within the realm of NLP and stands as one of the earliest quandaries in computational linguistics.

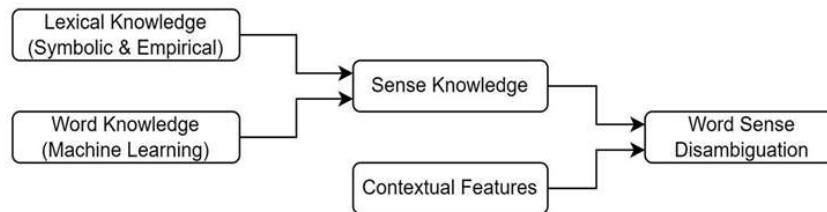


Figure 1. Conceptual Diagram of WSD.

WSD can be categorized into two classifications: “all words WSD” and “target word WSD”. In the case of all words WSD, the disambiguation process extends to all the words present in a given sentence, whereas target word WSD specifically focuses on disambiguating the target word within the sentence. WSD poses a significant challenge within the field of NLP and remains an ongoing area of research. It is regarded as an open problem, categorized as “AI-Complete”, signifying that a viable solution does exist but has not yet been discovered.

If we consider the following two sentences in the Hindi language:

आज-कल बाज़ार में नई-नई वस्तुओं की माँग बढ़ रही है। (aaj-kal baazaar mein naee-naee vastuon kee maang badh rahee hai)

(Now-a-days the demand of new things is increasing in the market.)

सुहागन औरतें अपनी माँग में सिंदूर भरती हैं। (suhaagan auraten apanee maang mein sindoor bharatee hain)

(Married women apply vermillion on their maang (the partition of hair on head).)

In both sentences, we have a common word, “माँग” (maang), that has a different meaning as per the context. In the initial sentence, the term refers to “the demand”, whereas in the subsequent sentence, it denotes “the partition of hair on the head”. Identifying the specific interpretation of a polysemous word is not a problem for a personage, whereas, for machines, it is a challenging task. Conversely, Hindi is the top fourth language, with over 615 million speakers worldwide.

2. Various Approaches for WSD

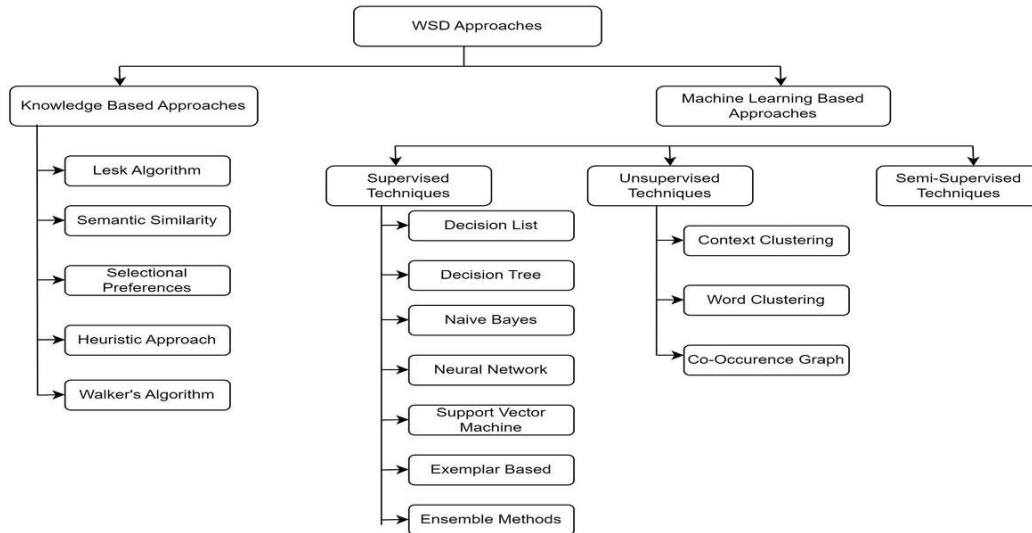


Figure 2. Classification of WSD Approaches.

3. Text Preprocessing

1. Lowercasing-

Converts all text to lowercase ("The" → "the").

Helps in reducing redundancy and avoids treating "Apple" and "apple" as different tokens.

Simple yet **crucial** for normalization.

```
df['review'] = df['review'].str.lower()
```

```
df['review'].str.lower().head()
```

```
0    one of the other reviewers has mentioned that ...
1    a wonderful little production. <br /><br />the...
2    i thought this was a wonderful way to spend ti...
3    basically there's a family where a little boy ...
4    petter mattei's "love in the time of money" is...
Name: review, dtype: object
```

2. Removing HTML Tags

- Often scraped data contains HTML.
- Use regex or libraries like `BeautifulSoup` to strip tags.

```
def remove_html_tags(text):
    pattern = re.compile('<.*?>')
    return pattern.sub(r'', text)
```

```
df['review'].apply(remove_html_tags).head()
```

```
0    one of the other reviewers has mentioned that ...
1    a wonderful little production. the filming tec...
2    i thought this was a wonderful way to spend ti...
3    basically there's a family where a little boy ...
4    petter mattei's "love in the time of money" is...
Name: review, dtype: object
```

```
df['review'] = df['review'].apply(remove_html_tags)
```

3. Removing URLs

- URLs rarely add value in most NLP tasks (unless URL itself is the focus).

```

def remove_urls(text):
    pattern = re.compile(r'https?://\S+|www\.\S+')
    return pattern.sub('', text)

text = "hello please visit https://github.com/deepankargupta856 to
view my projects you can google me as well www.google.com "

remove_urls(text)

{"type": "string"}

df['review'].apply(remove_urls).head()

0    one of the other reviewers has mentioned that ...
1    a wonderful little production. the filming tec...
2    i thought this was a wonderful way to spend ti...
3    basically there's a family where a little boy ...
4    petter mattei's "love in the time of money" is...
Name: review, dtype: object

df['review'] = df['review'].apply(remove_urls)

```

4. Removing Punctuation

- Keeps words consistent (e.g., "hello!" and "hello" are treated the same).
- Punctuation usually doesn't hold semantic meaning for standard NLP.

```

exclude = string.punctuation
print(exclude)#all punctuations from python

!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~

def self_punc_removal(text):
    for punc in exclude:
        text = text.replace(punc, '')
    return text

text = 'Hello! how are you doing? '
start = time.time()
print(self_punc_removal(text))
self_time = time.time() - start

Hello how are you doing

def auto_remove_punc(text):
    return text.translate(str.maketrans('', '', exclude))

start = time.time()
print(auto_remove_punc(text))
auto_time = time.time() - start

Hello how are you doing

self_time/auto_time##self method v/s auto method time taken
comparision

4.768025078369906

df['review'].apply(auto_remove_punc).head()

0    one of the other reviewers has mentioned that ...
1    a wonderful little production the filming tech...
2    i thought this was a wonderful way to spend ti...
3    basically theres a family where a little boy j...
4    petter matteis love in the time of money is a ...
Name: review, dtype: object

df['review'] = df['review'].apply(auto_remove_punc)

```

5. Chat-word / Slang Conversion

- Converts abbreviations or slang to formal equivalents:
 - IMHO → in my humble opinion, LOL → laughing out loud
- Useful for social media, chat, and informal text.


```
def chat_words_conv(text):
    transformed = []
    for w in text.split():
        if w.upper() in chat_words:
            transformed.append(chat_words[w.upper()])
        else:
            transformed.append(w)
    return ' '.join(transformed)

chat_words_conv('FYI I used to live in jaipur')

{"type": "string"}

df['review'] = df['review'].apply(chat_words_conv)
```

6. Spelling Error Handling

- Corrects misspelled words so "happee" and "happy" aren't treated differently.
- Can use:
 - TextBlob().correct()
 - SymSpell
 - Hunspell
 - SpellChecker (fastest but needs proper setup)

```
from textblob import TextBlob
```

```
incorrect_text = 'helloe my firend how do yu do?'
textblb = TextBlob(incorrect_text)
textblb.correct().string

{"type": "string"}

df['review'] = df['review'].apply(lambda x :
    str(TextBlob(x).correct()))## notoriously slow as because it checks
each word's spelling using a naive probabilistic model.

from spellchecker import SpellChecker

spell = SpellChecker()

def fast_correct(text):
    words = text.split()
    corrected = [spell.correction(w) or w for w in words]
    return ' '.join(corrected)

df['review'] = df['review'].apply(fast_correct)
```

4. WSD Execution Process

A string with an ambiguous word is given as an input string. Then, pre-processing is performed on this input string. Pre-processing steps such as stop word elimination, tokenization, part-of-speech tagging, and lemmatization, etc., are essential to transform raw text into a suitable format for analysis. For example, we have input ‘राम कच्चा आम खा रहा है।’ (raam kachcha aam kha raha hai) (Ram is eating raw mango). Various pre-processing steps are as follows:

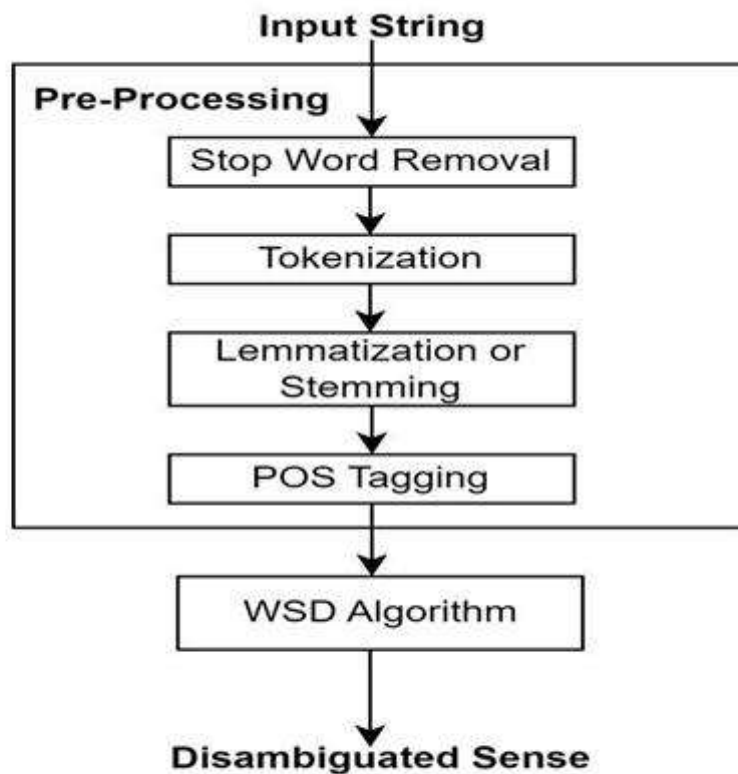
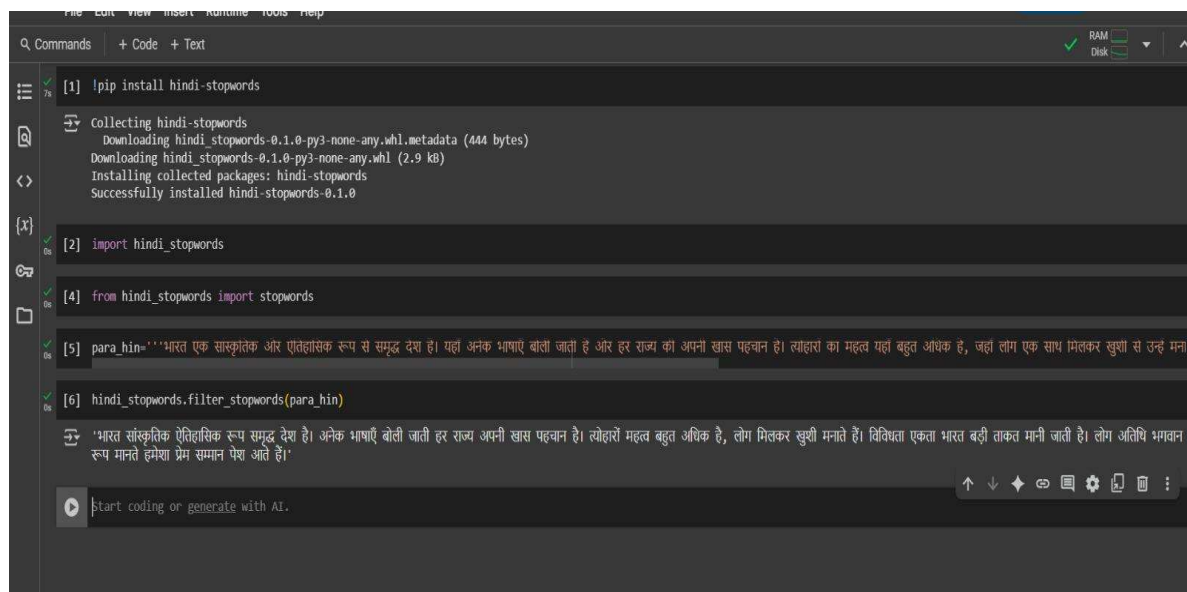


Figure 6. Flowchart of WSD Execution Process.

Stop Word Elimination: Stop words are words commonly filtered out or excluded from the analysis process in NLP. These words are highly frequent in most texts, but they generally lack significant meaning or do not contribute much to the overall understanding of the content. By eliminating stop words, the text becomes less noisy, and contextual relevance is improved. This improved context helps the WSD algorithm make more accurate sense selections. Examples of stop words in English include “the”, “a”, “an”, “in”, “on”, “at”, “and”, “but”, “or”, “I”, “you”, “he”, “she”, “it”, etc. Examples of stop words in Hindi (Devanagari script) include “का,” “की,” “के,” “को,” “है,” “हैं,” “में,” “और,” “लेकिन,” “या,” “मैं,” “तुम,” “वह,” “यह,” आदि।

The elimination of stop words and punctuation from the input text is performed in this step as they hold no significance or utility. After stop word removal string is ‘राम कच्चा आम खा’.

Instead of writing own stopwords as input, We can now directly import hindi stopwords which was earlier not possible.



```
File Edit View Insert Runtime Tools Help
Q Commands + Code + Text
[1] !pip install hindi-stopwords
Collecting hindi-stopwords
  Downloading hindi_stopwords-0.1.0-py3-none-any.whl.metadata (444 bytes)
  Downloading hindi_stopwords-0.1.0-py3-none-any.whl (2.9 kB)
  Installing collected packages: hindi-stopwords
  Successfully installed hindi-stopwords-0.1.0
[2] import hindi_stopwords
[4] from hindi_stopwords import stopwords
[5] para_hin="भारत एक सांस्कृतिक और ऐतिहासिक रूप से समृद्ध देश है। यहाँ अनेक भाषाएँ बोली जाती हैं और हर राज्य की अपनी खास पहचान है। लोहारों का महत्व यहाँ बहुत अधिक है, जहाँ लोग एक साथ मिलकर खुशी से उन्हें मनाते हैं।"
[6] hindi_stopwords.filter_stopwords(para_hin)
'भारत सांस्कृतिक ऐतिहासिक रूप से समृद्ध देश है। अनेक भाषाएँ बोली जाती हैं हर राज्य अपनी खास पहचान है। लोहारों महत्व बहुत अधिक है, लोग मिलकर खुशी मनाते हैं। विविधता एकता भारत बड़ी ताकत मानी जाती है। लोग अतिथि भगवान रूप मानते हमेशा प्रेम समान पेश आते हैं।'
```

Implementation-

```
from nltk.corpus import stopwords
nltk.download('stopwords')
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
eng_stop_words = set(stopwords.words('english'))
def remove_stopwords(text):
    transformed = []
    for w in text.split():
        if w not in eng_stop_words:
            transformed.append(w)
    return ' '.join(transformed)
df['review'][0]
{"type": "string"}
remove_stopwords(df['review'][0])
{"type": "string"}
df['review'] = df['review'].apply(remove_stopwords)
```

Tokenization: Tokenization is a fundamental technique in NLP that involves dividing a given text into smaller components, such as sentences and words. It encompasses the method of breaking down a string into a list of individual units called tokens. It helps in isolating individual words for disambiguation, making the WSD process more manageable and focused. In this context, a token can refer to a word within a sentence or a sentence within a paragraph, representing a fragment of the whole text. After Tokenization output is (‘राम’, ‘कच्चा’, ‘आम’, ‘खा’।).

Implementation-

```
sentences= nltk.sent_tokenize(paragraph)

sentences

[' My message, especially to young people, is to have courage to think
differently, courage to invent, to travel the unexplored path, courage
to discover the impossible, and to conquer the problems and succeed.',
 'These are great qualities that they must work towards.',
 'This is my message to the young people.',
 'You have to dream before your dreams can come true.',
 'Great dreams of great dreamers are always transcended.',
 'Thinking should become your capital asset, no matter whatever ups
and downs you come across in your life.',
 'The ignited minds of the youth are the most powerful resource on the
earth, above the earth, under the earth.']]

type(sentences)

list

for i in range (len(sentences)):
    words= nltk.word_tokenize(sentences[i])
    words=[stemmer.stem(word) for word in words if word not in
set(stopwords.words('english'))]
    sentences[i]=' '.join(words)

sentences
```

Output-

```
['my messag , especy young peopl , courag think differ , courag invent
, travel unexplor path , courag discov imposs , conquer problem
succeed .',
 'these great qualiti must work toward .',
 'thi messag young peopl .',
 'you dream dream come true .',
 'great dream great dreamer alway transcend .',
 'think becom capit asset , matter whatev up down come across life .',
 'the ignit mind youth power resourc earth , earth , earth .']]
```

Stemming: Stemming is a linguistic process aimed at removing the last few characters of a word, which can sometimes result in incorrect meanings and altered spellings. Stemming simplifies text data and improves computational efficiency, aiding in tasks such as text matching and retrieval. However, it may generate non-words, leading to potential loss of word meaning and semantic distinctions. For instance, stemming the word 'Caring' would return to 'Car', which is an incorrect result.

Implementation-

```
from nltk.stem import PorterStemmer

stemming= PorterStemmer()

text="""Every individual carries within them an untapped potential,
waiting to be discovered. Life will always present challenges –
moments when you doubt your path, your strength, or your worth. But it
is in those moments that true growth begins. Do not wait for the
perfect opportunity. Create it. Take small steps every day towards
your goals, and those steps will become a journey. Believe in your
ideas, even when others don't. Your consistency and passion are more
powerful than instant success. The world doesn't need perfection; it
needs people who are real, resilient, and ready to learn. So rise,
work hard, and never stop evolving."""

words = text.split()

for word in words:
    print(word + " ---> " + stemming.stem(word))

Every ---> everi
individual ---> individu
carries ---> carri
within ---> within
them ---> them
an ---> an
untapped ---> untap
potential, ---> potential,
waiting ---> wait
to ---> to
be ---> be
discovered. ---> discovered.
Life ---> life
will ---> will
always ---> alway
```

Lemmatization: Lemmatization takes into account the context of a word and transforms it into its meaningful base form, known as a lemma. For example, by lemmatizing the word 'Caring,' the resulting lemma would be 'Care', which is the correct result. By converting words to their lemma, the WSD system can associate different inflected forms of a word with the same sense, improving the coverage and generalization of the sense inventory.

```
from nltk.stem import WordNetLemmatizer

lemmatizer= WordNetLemmatizer()

lemmatizer.lemmatize("going")
'going'

lemmatizer.lemmatize("going",pos='v')
'go'

for word in words:
    print(word+"---->"+lemmatizer.lemmatize(word))

Every---->Every
individual---->individual
carries---->carry
within---->within
them---->them
an---->an
untapped---->untapped
potential,---->potential,
waiting---->waiting
to---->to
be---->be
discovered.---->discovered.
Life---->Life
will---->will
always---->always
present---->present
challenges---->challenge
----->—
moments---->moment
when---->when
you---->you
```


Stemming vs Lemmatization

Feature	Stemming	Lemmatization
Definition	Removes suffixes to get the root form	Reduces word to its base or dictionary form
Output	May not be a real word	Always a valid word
Method	Rule-based (Porter, Snowball, Lancaster)	Dictionary + PoS based (WordNet, spaCy)
Speed	Fast	Slower (due to dictionary lookup)
Accuracy	Lower – crude chopping	Higher – context-aware
Language Knowledge	None	Requires knowledge of grammar and word meaning
Examples	"studies" → "studi" "fishing" → "fish"	"studies" → "study" "fishing" → "fish"
When to Use	Quick & dirty NLP tasks, search engines	Production NLP, user-facing apps, text analytics
Tool Support	NLTK, Snowball, Regex	NLTK (WordNet), spaCy

```
stemmer= PorterStemmer()
import nltk
nltk.download('punkt_tab')
[nltk_data] Downloading package punkt_tab to
[nltk_data] C:\Users\Acer\AppData\Roaming\nltk_data...
[nltk_data] Package punkt_tab is already up-to-date!
True
sentences= nltk.sent_tokenize(paragraph)
sentences
[' My message, especially to young people, is to have courage to think
differently, courage to invent, to travel the unexplored path, courage
to discover the impossible, and to conquer the problems and succeed.',
'These are great qualities that they must work towards.',
'This is my message to the young people.',
'You have to dream before your dreams can come true.',
'Great dreams of great dreamers are always transcended.',
'Thinking should become your capital asset, no matter whatever ups
and downs you come across in your life.',
'The ignited minds of the youth are the most powerful resource on the
earth, above the earth, under the earth.']
type(sentences)
list
for i in range (len(sentences)):
    words= nltk.word_tokenize(sentences[i])
    words=[stemmer.stem(word) for word in words if word not in
set(stopwords.words('english'))]
    sentences[i]=' '.join(words)
sentences
```



```

['my messag , especí young peopl , courag think differ , courag invent
, travel unexplor path , courag discov imposs , conquer problem
succeed .',
'these great qualiti must work toward .',
'thi messag young peopl .',
'you dream dream come true .',
'great dream great dreamer alway transcend .',
'think becom capit asset , matter whatev up down come across life .',
'the ignit mind youth power resourc earth , earth , earth .']

# steeming not give good result
from nltk.stem import SnowballStemmer
snowballstemmer= SnowballStemmer('english')
for i in range (len(sentences)):
    words= nltk.word_tokenize(sentences[i])
    words=[snowballstemmer.stem(word) for word in words if word not in
set(stopwords.words('english'))]
    sentences[i]=' '.join(words)

sentences

['my messag , especí young peopl , courag think differ , courag invent
, travel unexplor path , courag discov imposs , conquer problem
succeed .',
'these great qualiti must work toward .',
'this messag young peopl .',
'you dream dream come true .',
'great dream great dreamer alway transcend .',
'think becom capit asset , matter whatev up down come across life .',
'the ignit mind youth power resourc earth , earth , earth .']

# by snowball-->
# all letter are smalland for some words its not giving god result

# LEMMETIZATION
from nltk.stem import WordNetLemmatizer
lemmatizer= WordNetLemmatizer()
for i in range (len(sentences)):
    words= nltk.word_tokenize(sentences[i])
    words=[lemmatizer.lemmatize(word,pos='v') for word in words if
word not in set(stopwords.words('english'))]
    sentences[i]=' '.join(words)

sentences

```

```

['My message , especially young people , courage think differently ,
courage invent , travel unexplored path , courage discover
impossible , conquer problems succeed .',
'These great qualities must work towards .',
'This message young people .',
'You dream dream come true .',
'Great dream great dreamers always transcend .',
'Thinking become capital asset , matter whatever up down come across
life .',
'The ignite mind youth powerful resource earth , earth , earth .']

# all stopwords got deleted getting better result

```

PoS Tagging: POS tagging involves the assignment of suitable part-of-speech labels to each word within a sentence, encompassing categories such as nouns, adverbs, verbs, pronouns, adjectives, conjunctions, and their respective sub-categories. This information is crucial for WSD because different parts of speech may have different senses. POS tagging helps in narrowing down the sense options for each word based on its grammatical role in the sentence.

```
sent= "On July 20, 1969, Neil Armstrong became the first human to set
foot on the Moon during NASA's Apollo 11 mission. Alongside Buzz
Aldrin and Michael Collins, he launched from Kennedy Space Center in
Florida. The mission was a defining moment in the space race between
the United States and the Soviet Union. Later, in 1999, Armstrong gave
a lecture at the Massachusetts Institute of Technology (MIT),
highlighting the efforts of scientists and engineers at NASA. His
legacy continues to inspire young astronauts and researchers across
the world."

import nltk
words=nltk.word_tokenize(sent)
print(words)

['On', 'July', '20', ',', '1969', ',', 'Neil', 'Armstrong', 'became',
'the', 'first', 'human', 'to', 'set', 'foot', 'on', 'the', 'Moon',
'during', 'NASA', "'s", 'Apollo', '11', 'mission', '.', 'Alongside',
'Buzz', 'Aldrin', 'and', 'Michael', 'Collins', 'he', 'launched',
'from', 'Kennedy', 'Space', 'Center', 'in', 'Florida', '.', 'The',
'mission', 'was', 'a', 'defining', 'moment', 'in', 'the', 'space',
'race', 'between', 'the', 'United', 'States', 'and', 'the', 'Soviet',
'Union', '.', 'Later', 'in', '1999', 'Armstrong', 'gave',
'a', 'lecture', 'at', 'the', 'Massachusetts', 'Institute', 'of',
'Technology', '(', 'MIT', ')', 'highlighting', 'the', 'efforts',
'of', 'scientists', 'and', 'engineers', 'at', 'NASA', '.', 'His',
'legacy', 'continues', 'to', 'inspire', 'young', 'astronauts', 'and',
'researchers', 'across', 'the', 'world', '.']

import nltk
nltk.download('averaged_perceptron_tagger_eng')

tag_elements= nltk.pos_tag(words)
print(tag_elements)

# GOT PARTS OF SPEECH TAG

[('On', 'IN'), ('July', 'NNP'), ('20', 'CD'), (',', ','), ('1969',
'CD'), (',', ','), ('Neil', 'NNP'), ('Armstrong', 'NNP'), ('became',
'VBD'), ('the', 'DT'), ('first', 'JJ'), ('human', 'JJ'), ('to', 'TO'),
('set', 'VB'), ('foot', 'NN'), ('on', 'IN'), ('the', 'DT'), ('Moon',
'NNP'), ('during', 'IN'), ('NASA', 'NNP'), ("'", 'POS'), ('Apollo',
'NNP'), ('11', 'CD'), ('mission', 'NN'), ('.', '.'), ('Alongside',
'NNP'), ('Buzz', 'NNP'), ('Aldrin', 'NNP'), ('and', 'CC'), ('Michael',
'NNP'), ('Collins', 'NNP'), (',', ','), ('he', 'PRP'), ('launched',
'VBD'), ('from', 'IN'), ('Kennedy', 'NNP'), ('Space', 'NNP'),
('Center', 'NNP'), ('in', 'IN'), ('Florida', 'NNP'), ('.', '.'),
('The', 'DT'), ('mission', 'NN'), ('was', 'VBD'), ('a', 'DT'),
('defining', 'JJ'), ('moment', 'NN'), ('in', 'IN'), ('the', 'DT'),
('space', 'NN'), ('race', 'NN'), ('between', 'IN'), ('the', 'DT'),
('United', 'NNP'), ('States', 'NNPS'), ('and', 'CC'), ('the', 'DT'),
('Soviet', 'NNP'), ('Union', 'NNP'), ('.', '.'), ('Later', 'NNP'),
(',', ','), ('in', 'IN'), ('1999', 'CD'), (',', ','), ('Armstrong',
'NNP'), ('gave', 'VBD'), ('a', 'DT'), ('lecture', 'NN'), ('at', 'IN'),
('the', 'DT'), ('Massachusetts', 'NNP'), ('Institute', 'NNP'), ('of',
'IN'), ('Technology', 'NNP'), ('(', '('), ('MIT', 'NNP'), (',', ','),
(')', ')'), ('highlighting', 'VBG'), ('the', 'DT'), ('efforts',
'NNS'), ('of', 'IN'), ('scientists', 'NNS'), ('and', 'CC'),
('engineers', 'NNS'), ('at', 'IN'), ('NASA', 'NNP'), ('.', '.'),
('His', 'PRP$'), ('legacy', 'NN'), ('continues', 'VBZ'), ('to', 'TO'),
('inspire', 'VB'), ('young', 'JJ'), ('astronauts', 'NNS'), ('and',
'CC'), ('researchers', 'NNS'), ('across', 'IN'), ('the', 'DT'),
('world', 'NN'), ('.', '.')]

```

Word2Vec

Computers can't understand language like humans do — they need **numbers**, not letters. So we use Word2Vec to:

- Represent **semantic meaning** of words as **vectors**
- Allow machines to **compare**, **cluster**, or **search** for words based on meaning
- Enable models to learn that:
 - "king" is related to "queen" like "man" is to "woman"
 - "school" is more similar to "college" than "banana"

Operations as such can also be performed on Vectors-

```
[11]: vec= wv['king']-wv['man']+wv['woman']
```

```
[12]: vec
```

```
[12]: array([ 0.41736597,  0.90427005, -1.0050299, -0.06202102,  0.49725997,
            0.80667007, -0.14855,   0.80365,   -0.15653998, -0.66973996,
            0.23435399,  0.62476,   0.925871,  -0.97099996,  0.92566,
            0.89915,   -1.54596,  -0.52625,   0.13695401,  0.66199005,
            0.4871601,  0.37035,  -0.214214,  0.10100996,  0.71358,
           -2.0874999, -1.1362001, -1.1496099, -0.53599,   0.27389997,
            1.6723,    0.02930999, -0.77656007,  0.46056286,  0.34866,
           -0.05741701,  0.19444,  -0.207748,  -0.73038995, -0.10751998,
            0.235544,   0.96423995, -0.46993998, -0.48727497, -0.25399995,
            0.4621299, -0.66081,   -1.9451499, -0.68797004, -0.49784005],
          dtype=float32)
```

```
[13]: wv.most_similar([vec])
```

```
[13]: [('king', 0.8859834671020508),
      ('queen', 0.8609582185745239),
      ('daughter', 0.7684512734413147),
      ('prince', 0.7640699744224548),
      ('throne', 0.7634970545768738),
      ('princess', 0.7512729167938232),
      ('elizabeth', 0.7506489157676697),
      ('father', 0.7314497232437134),
      ('kingdom', 0.7296158671379089),
      ('mother', 0.728001058101654)]
```

Implementation Process-

1. Tokenize the sentence

Split the sentence into individual words:

```
sentence = "राम स्कूल जाता है"
words = sentence.split()
```

```
Vocabulary = ["राम", "स्कूल", "जाता", "है"]
```

We then create a **mapping**:

- `word_to_index` → maps each word to a unique index.
- `index_to_word` → maps each index back to the word.

2. Generate training data (Skip-gram)

With a **window size = 1**, we generate (center, context) pairs.

For example:

Words: राम स्कूल जाता है
Pairs: (राम, स्कूल), (स्कूल, राम), (स्कूल, जाता), (जाता, स्कूल), (जाता, है), (है, जाता)

Each pair is a training example:

- Input = center word
- Output = context word

3. One-hot Encoding

We represent each word as a one-hot vector (vector with all 0s except a 1 at the index of the word).

For example: If vocabulary size is 4, then "राम" might be $[1, 0, 0, 0]$

4. Model Architecture

This is a **very simple neural network**:

- Input layer: one-hot encoded word vector
- Hidden layer: gives the **word embedding**
- Output layer: predicts the context word probabilities via **softmax**

We use two weight matrices:

- W_1 : from input to embedding
- W_2 : from embedding to output layer

5. Training (Forward & Backpropagation)

- **Forward pass:**
 - Input word → hidden layer (embedding)
 - Multiply with second matrix → output logits
 - Apply softmax → prediction

- **Loss function:**
 - Cross-entropy between predicted and true context word
- **Backpropagation:**
 - Gradients are computed and weights are updated via gradient descent.

This is done for multiple epochs.

6. Final Word Embeddings

After training, the matrix W_1 contains **embedding vectors** for each word.

For example:

राम: [0.21, 0.81, 0.37, 0.04, 0.92]

स्कूल: [0.47, 0.19, 0.71, 0.36, 0.51]

Words used in **similar contexts** will have **similar vectors**.

Implementation-

```

from gensim.models import Word2Vec

# Example tokenized Hindi sentences
sentences = [
    ["भारत", "एक", "महान", "देश", "है"],
    ["हम", "सब", "भारतीय", "हैं"],
    ["यहाँ", "की", "संस्कृति", "बहुत", "पुरानी", "है"]
]

# Train Word2Vec model
model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)

# Save model
model.save("hindi_word2vec.model")

# Load model and get word vector
print("Vector for 'भारत':\n", model.wv["भारत"])

```

Vector for 'भारत':

```

[-0.00950012  0.00956222 -0.00777076 -0.00264551 -0.00490641 -0.0049667
 -0.00802359 -0.00778358 -0.00455321 -0.00127536 -0.00510299  0.00614054
 -0.00951662 -0.0053071  0.00943715  0.00699133  0.00767582  0.00423474
  0.00050709 -0.00598114  0.00601878  0.00263503  0.00769943  0.00639384
  0.00794257  0.00865741 -0.00989575 -0.0067557  0.00133757  0.0064403
  0.00737382  0.00551698  0.00766163 -0.00512557  0.00658441 -0.00410837
 -0.00905534  0.00914168  0.0013314  -0.00275968 -0.00247784 -0.00422048
  0.00481234  0.00440022 -0.00265336 -0.00734188 -0.00356585 -0.00033661
  0.00609589 -0.00283734 -0.00012089  0.00087973 -0.00709565  0.002065
 -0.00143242  0.00280215  0.00484222 -0.00135202 -0.00278014  0.00773865
  0.0050456  0.00671352  0.00451564  0.00866716  0.00747497 -0.00108189
  0.00874764  0.00460172  0.00544063 -0.00138608 -0.00204132 -0.00442435
 -0.0085152  0.00303773  0.00888319  0.00891974 -0.00194235  0.00608616
  0.00377972 -0.00429597  0.00204292 -0.00543789  0.00820889  0.00543291
  0.00318443  0.00410257  0.00865715  0.00727203 -0.00083347 -0.00707277
  0.00838047  0.00723358  0.00173047 -0.00134749 -0.00589009 -0.00453309
  0.00864797 -0.00313511 -0.00633882  0.00987008]

```



```

from gensim.models import KeyedVectors

# Load the Hindi FastText word vectors
model = KeyedVectors.load_word2vec_format('cc.hi.300.vec', binary=False)

# Test the model
print("Vector for भारत:\n", model["भारत"])

```

Vector for भारत:

```

[ 2.530e-02 -4.500e-03 -2.860e-02 -2.900e-02 -2.600e-02  2.300e-03
 3.540e-02  2.500e-02 -2.310e-02  1.300e-03  2.870e-02 -5.100e-03
 1.320e-02 -2.000e-02 -8.000e-04  1.880e-02  7.670e-02  1.820e-02
 1.350e-02  2.234e-01 -2.210e-02 -1.180e-02  1.870e-02  1.560e-02
-4.460e-02 -2.700e-02  3.260e-02 -7.090e-02 -2.200e-02  2.740e-02
 2.440e-02 -1.380e-02  3.420e-02  1.910e-02 -3.890e-02 -3.980e-02
-2.020e-02 -5.100e-03 -2.500e-02  4.980e-02 -2.000e-03 -7.700e-03
 6.100e-03 -6.500e-03 -4.540e-02  1.455e-01 -1.620e-02 -2.500e-02
-3.220e-02  2.760e-02 -2.910e-02 -2.830e-02 -2.390e-02  3.270e-02
-2.190e-02  1.000e-03  3.520e-02  1.900e-03  2.800e-02 -4.600e-03
-5.680e-02 -9.050e-02 -3.000e-04  2.570e-02  2.130e-02 -7.220e-02
 2.340e-02 -5.420e-02  1.520e-02 -2.880e-02 -1.090e-02 -8.940e-02
 2.740e-02 -5.010e-02 -3.700e-02  1.170e-02 -1.230e-02 -1.000e-02
 1.150e-02  2.570e-02  1.200e-03 -1.650e-02  2.350e-02  9.000e-03
 1.000e-04 -1.770e-02  6.700e-03  1.510e-02  9.950e-02 -1.822e-01
-2.720e-02 -4.610e-02 -1.710e-02  3.920e-02 -2.180e-02 -5.800e-03
 3.150e-02 -5.490e-02  1.570e-02  1.960e-02 -1.960e-02  3.030e-02
 7.600e-03 -1.170e-02 -1.670e-02 -1.250e-02  3.800e-03 -2.950e-02
 6.980e-02  3.480e-02 -2.030e-02 -1.090e-02  1.460e-02 -5.920e-02
 4.920e-02  2.770e-02 -3.440e-02 -2.020e-02  8.100e-02 -4.600e-03
-4.800e-02  8.440e-02  8.800e-03  3.360e-02  7.000e-04  7.240e-02
-2.950e-02 -1.348e-01  2.030e-02  5.500e-03  7.800e-03 -2.570e-02
-3.440e-02  3.710e-02  4.600e-02  3.690e-02  1.960e-02 -3.780e-02
-1.700e-03  2.130e-02  1.290e-02 -4.460e-02 -4.500e-03  1.110e-02
-7.500e-03  7.500e-03 -7.000e-03 -3.490e-02  9.900e-03  2.120e-02
-2.960e-02  1.020e-02  2.630e-02 -1.044e-01 -6.400e-03  2.130e-02
-5.800e-03  6.600e-03 -2.130e-02  4.670e-02 -4.030e-02 -1.080e-01
-1.370e-02 -1.880e-02  3.110e-02 -1.700e-02  1.160e-02  1.370e-02
-1.070e-02  1.920e-02 -2.500e-03 -2.790e-02 -1.466e-01  6.980e-02
-1.140e-02  3.600e-03  3.500e-03 -6.660e-02 -2.450e-02  1.723e-01
 3.090e-02  5.180e-02  5.840e-02  8.300e-03 -4.030e-02 -3.430e-02

```

Generating similar Words Using Vectors-

```
similar_words = model.most_similar("भारत", topn=10)
for word, similarity in similar_words:
    print(f"{word}: {similarity:.4f}")
```

देश: 0.6862
भारतीय: 0.6340
पाकिस्तान: 0.6237
राज्य: 0.6183
विश्व: 0.5960
चीन: 0.5896
प्रदेश: 0.5876
अमेरिका: 0.5865
ब्रिटिश: 0.5657
दक्षिण: 0.5628

```
similar_words = model.most_similar("शिक्षा", topn=10)
for word, similarity in similar_words:
    print(f"{word}: {similarity:.4f}")
```

शिक्षाशिक्षा: 0.6962
है.शिक्षा: 0.6750
शिक्षा12: 0.6745
मा.शिक्षा: 0.6712
खंडशिक्षा: 0.6706
यौनशिक्षा: 0.6664
जनशिक्षा: 0.6619
कीशिक्षा: 0.6614
स्कूलशिक्षा: 0.6607
राजशिक्षा: 0.6606

Naïve bayes Classifier

Implementation Steps-

1. Prepare a labeled dataset

You need training examples of the **ambiguous word** used in **different senses**:

Sentence	Sense Label
"कल स्कूल गया था।"	Past (Yesterday)
"कल का इम्तिहान मुश्किल है।"	Future (Tomorrow)
"कल बहुत सुंदर था।"	Past
"कल सुबह पढ़ाई करनी है।"	Future

You can manually label or use Hindi WordNet data (if available).

2. Preprocess the Text

Use Indic NLP or spaCy Hindi models to:

- Tokenize
- Remove stop words (optional)
- Normalize words (optional)

```
tokens = indic_tokenize.trivial_tokenize(sentence, lang='hi')
```

3. Feature Extraction

Convert context into features:

- Surrounding words (left/right of ambiguous word)
- Bag of Words (BoW)
- POS tags (optional)

Example features:

Sentence: "कल का इम्तिहान मुश्किल है।"

Features: ['का', 'इम्तिहान', 'मुश्किल', 'है']

Use `CountVectorizer` or `TfidfVectorizer` to convert text to numeric vectors.

4. Train Naive Bayes Classifier

Use `MultinomialNB` from `sklearn`:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(all_sentences) # sentence context
y = sense_labels # e.g., ['past', 'future', 'past', ...]

model = MultinomialNB()
model.fit(X, y)
```

5. Predict the Sense of New Sentences

```
new_sentence = "कल ऑफिस जाऊँगा।"
X_test = vectorizer.transform([new_sentence])
predicted_sense = model.predict(X_test)
print(predicted_sense)
```

Taking the initial Example which we started with-

```
train_vakya = [  
    ("आजकल तकनीक की माँग हर क्षेत्र में बढ़ रही है", "demand"),  
    ("विवाहित महिलाएँ त्योहारों पर अपनी माँग में सिंदूर लगाती हैं।", "parting"),  
    ("सरकार से वेतन वृद्धि की माँग की जा रही है", "demand"),  
    ("उसकी माँग में सिंदूर देखकर सब समझ गए", "parting"),  
    ("बाजार में इस उत्पाद की माँग बहुत अधिक है", "demand"),  
    ("दुल्हन की माँग बहुत सुंदर लग रही थी", "parting")  
]
```

```
featureset = [(extract_features(sent), label) for (sent, label) in train_vakya]
```

```
classifier = NaiveBayesClassifier.train(featureset)
```

```
test_sentences = [  
    "सरकार ने किसानों की माँग को मान लिया",  
    "उसने अपनी माँग में लाल सिंदूर भरा",  
    "शादी के बाद माँग भरना शुभ माना जाता है",  
    "त्योहार के समय बिजली की माँग बढ़ जाती है"  
]  
  
for sent in test_sentences:  
    features = extract_features(sent)  
    prediction = classifier.classify(features)  
    print(f"{sent} ➤ Sense: {prediction}")
```

सरकार ने किसानों की माँग को मान लिया ➤ Sense: demand

उसने अपनी माँग में लाल सिंदूर भरा ➤ Sense: parting

शादी के बाद माँग भरना शुभ माना जाता है ➤ Sense: demand

त्योहार के समय बिजली की माँग बढ़ जाती है ➤ Sense: demand

Another Example-

```
train_data = [  
    ("उसने बैंक में पैसे जमा किए", "financial"),  
    ("वे नदी के किनारे बैठे", "river"),  
    ("बैंक ने उसका लोन मंजूर कर दिया", "financial"),  
    ("बच्चे नदी के बैंक पर खेल रहे थे", "river")  
]
```

```
featuresets = [(extract_features(sent), label) for (sent, label) in train_data]
```

```
classifier = NaiveBayesClassifier.train(featuresets)
```

```
test_sentence = "बच्चे बैंक के पास खेल रहे थे"  
test_features = extract_features(test_sentence)
```

```
print("Sentence:", test_sentence)  
print("Predicted sense:", classifier.classify(test_features))
```

Sentence: बच्चे बैंक के पास खेल रहे थे
Predicted sense: river

```
examples = [  
    "बैंक में पैसे जमा करने गया",  
    "नदी के किनारे बैठकर वह गीत गा रही थी",  
    "उसने अपने अकाउंट से पैसे निकाले",  
    "बच्चे नदी के बैंक पर पत्थर फेंक रहे थे",  
    "बैंक ने उसका लोन अस्वीकार कर दिया",  
    "वे लोग नदी के बैंक से मछली पकड़ रहे थे"  
]  
  
for sent in examples:  
    features = extract_features(sent)  
    label = classifier.classify(features)  
    print(f"Sentence: {sent} ► Predicted Sense: {label}")
```

Sentence: बैंक में पैसे जमा करने गया ► Predicted Sense: financial
Sentence: नदी के किनारे बैठकर वह गीत गा रही थी ► Predicted Sense: river
Sentence: उसने अपने अकाउंट से पैसे निकाले ► Predicted Sense: financial
Sentence: बच्चे नदी के बैंक पर पत्थर फेंक रहे थे ► Predicted Sense: river
Sentence: बैंक ने उसका लोन अस्वीकार कर दिया ► Predicted Sense: financial
Sentence: वे लोग नदी के बैंक से मछली पकड़ रहे थे ► Predicted Sense: river