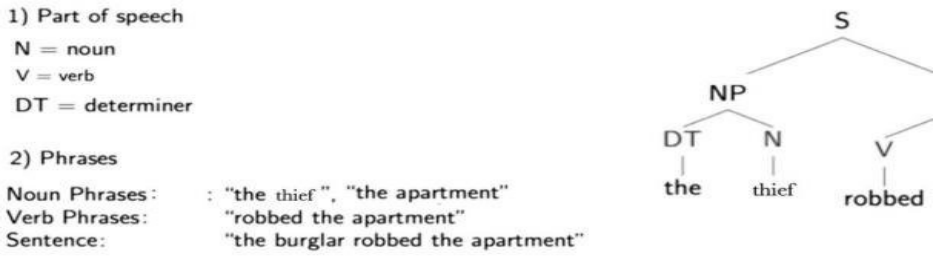
21CSE356T
NATURAL LANGUAGE PROCESSING
UNIT-1
OVERVIEW AND WORD LEVEL ANALYSIS

- Introduction to Natural Language Processing
- Applications of NLP
- Levels of NLP
- Regular Expressions
- Morphological Analysis
- Tokenization
- Stemming
- Lemmatization
- Feature extraction
    - Term Frequency (TF)
    - Inverse Document Frequency (IDF)
    - Modeling using TF-IDF
- Parts of Speech Tagging
- Named Entity Recognition
- N-grams
- Smoothing

INTRODUCTION TO NATURAL LANGUAGE PROCESSING

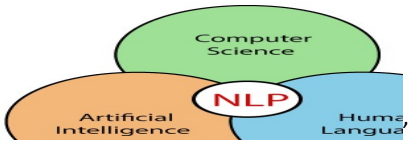
Language is a method of communication with the help of which we can speak, read and write. Natural Language Processing (NLP) is a subfield of Computer Science that deals with Artificial Intelligence (AI), which enables computers to understand and process human language.

Natural language processing (**NLP**) is the intersection of computer science, linguistics and machine learning. The field focuses on communication between computers and humans in natural language and **NLP** is all about making computers understand and generate human language.



- **Phonology** – This science helps to deal with patterns present in the sound and speeches related to the sound as a physical entity.

- **Pragmatics** – This science studies the different uses of language.
- **Morphology** – This science deals with the structure of the words and the systematic relations between them.
- **Syntax** – This science deal with the structure of the sentences.
- **Semantics** – This science deals with the literal meaning of the words, phrases as well as sentences.

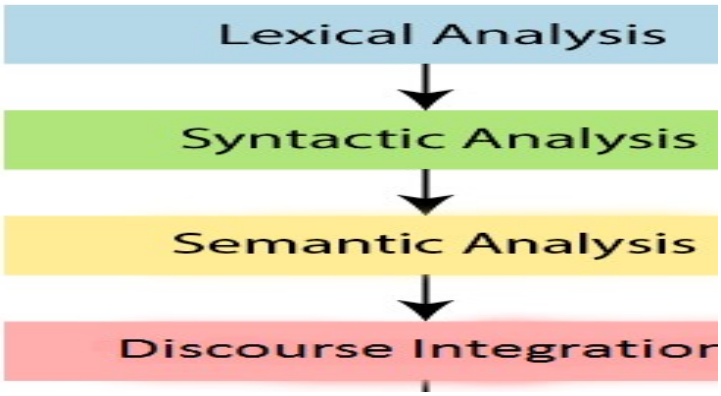


APPLICATIONS OF NLP

- **Search Autocorrect and Autocomplete:** Whenever you search something on Google, after typing 2-3 letters, it shows you the possible search terms.
- **Language Translator:** Have you ever used Google Translate to find out what a particular word or phrase is in a different language? I'm sure it's a YES!!
- **Social Media Monitoring:** More and more people these days have started using social media for posting their thoughts about a particular product, policy, or matter.
- **Chatbots:** Customer service and experience is the most important thing for any company.
- **Survey Analysis:** Surveys are an important way of evaluating a company's performance.
- **Targeted Advertising:** One day I was searching for a mobile phone on Amazon, and a few minutes later, Google started showing me ads related to similar mobile phones.
- **Hiring and Recruitment:** The Human Resource department selects the most important job of selecting the right employees for a company.
- **Voice Assistants:** Google Assistant, Apple Siri, Amazon Alexa
- **Grammar Checkers: M**ost widely used applications of natural language processing.
- **Email Filtering:** Have you ever used Gmail?
- **Sentiment Analysis:** Natural language understanding is particularly difficult for machines when it comes to opinions, given that humans often use sarcasm and irony.
- **Text Classification:** Text classification, a text analysis task that also includes sentiment analysis, involves automatically understanding, processing, and categorizing unstructured text.
- **Text Extraction:** Text extraction, or information extraction, automatically detects specific information in a text, such as names, companies, places, and more. This is also known as ***named entity recognition***.
- **Machine Translation:** Machine translation (MT) is one of the first applications of natural language processing.
- **Text Summarization:** There are two ways of using natural language processing to summarize data: *extraction-based summarization* – which extracts keyphrases and creates a summary, without adding any extra information – and *abstraction-based summarization,* which creates new phrases paraphrasing the original source.

- **Text-based applications:** Text-based applications involve the processing of written text, such as books, newspapers, reports, manuals, e-mail messages, and so on. These are all reading-based tasks. Text-based natural language research is ongoing in applications such as
  - finding appropriate documents on certain topics from a data-base of texts (for example, finding relevant books in a library)
  - extracting information from messages or articles on certain topics (for example, building a database of all stock transactions described in the news on a given day)
  - translating documents from one language to another (for example, producing automobile repair manuals in many different languages)
  - summarizing texts for certain purposes (for example, producing a 3-page summary of a 1000-page government report)
- **Dialogue-based applications** involve human-machine communication. Most naturally this involves spoken language, but it also includes interaction using keyboards. Typical potential applications include
  - question-answering systems, where natural language is used to query a database (for example, a query system to a personnel database)
  - automated customer service over the telephone (for example, to perform banking transactions or order items from a catalogue)
  - tutoring systems, where the machine interacts with a student (for example, an automated mathematics tutoring system)
  - spoken language control of a machine (for example, voice control of a VCR or computer)
  - general cooperative problem-solving systems (for example, a system that helps a person plan and schedule freight shipments)

LEVELS OF NLP



- **Lexical Analysis (Morphological Processing)**
  It is the first phase of NLP. The purpose of this phase is to break chunks of language input into sets of tokens corresponding to paragraphs, sentences and words. For example, a word like **"uneasy"** can be broken into two sub-word tokens as **"un-easy"**.
  - Lemmatization
  - Stopwords Removal
  - Correcting Misspelled Words
  - Morhpology
- **Syntactic Analysis (Parsing)**
  It is the second phase of NLP. The purpose of this phase is **two folds**: to check that a sentence is well formed or not and to break it up into a structure that shows the syntactic relationships between the different words. For example, the sentence like *"The school goes to the boy"* would be rejected by syntax analyzer or parser.
  - POS tagging

- **Semantic Analysis**

  It is the third phase of NLP. The purpose of this phase is to draw **exact meaning**, or you can say **dictionary meaning** from the text. The text is checked for meaningfulness. For example, semantic analyzer would reject a sentence like "*Hot ice-cream*".
  - Named Entity Recognition (NER)
  - Word Sense Disambiguation (WSD)
- **Discourse Integration**

  Discourse Integration depends upon the sentences that proceeds it and also invokes the meaning of the sentences that follow it.
  - Contextual Reference
  - Anaphora Resolution
- **Pragmatic Analysis**

  It is the last phase of NLP. Pragmatic analysis simply fits the actual objects/events, which exist in a given context with object references obtained during the last phase (semantic analysis). For example, the sentence *"Put the banana in the basket on the shelf"* can have two semantic interpretations and pragmatic analyzer will choose between these two possibilities.
  - Contextual Greeting
  - Figurative Expression

REGULAR EXPRESSIONS

- **Regex or Regular expression**, a language for specifying text search strings.
- Practical application is used in every computer language, in text processing tools like the Unix tools ***grep***, and in editors like ***vim or Emacs***.
- An algebraic notation for characterizing a set of strings.
- Regular expressions are particularly useful for searching in texts.

Disjunctions

- **Letters inside square brackets []**

Pattern	Matches
[wW]oodchuck	Woodchuck, woodchuck
[1234567890]	Any one digit

- **Ranges using the dash [A-Z]**

Pattern	Matches	Example
[A-Z]	An upper-case letter	Drenched Blossoms
[a-z]	A lower-case letter	my beans were impatient
[0-9]	A single digit	Chapter 1: Down the Rabbit Hole

Negation in Disjunction: Carat as first character in [] negates the list

Pattern	Matches	Examples
[^A-Z]	Not an upper case letter	Oyfn pripetchik
[^Ss]	Neither 'S' nor 's'	I have no exquisite reason"
[^.]	Not a period	Our resident Djinn
[e^]	Either e or ^	Look up ^ now

Convenient aliases

Pattern	Expansion	Matches	Examples
\d	[0-9]	Any digit	Fahreneit 451
\D	[^0-9]	Any non-digit	Blue Moon

\w	[a-ZA-Z0-9_]	Any alphanumeric or _	Daiyu
\W	[^\w]	Not alphanumeric or _	Look!
\s	[\r\t\n\f]	Whitespace (space, tab)	Look␣up
\S	[^\s]	Not whitespace	Look up

- **More Disjunction:** The pipe symbol | for disjunction

Pattern	Matches
groundhog|woodchuck	woodchuck
yours|mine	Yours
a|b|c	= [abc]
[gG]roundhog|[Ww]oodchuck	Woodchuck

- **Wildcards, optionality, repetition: . ? * +**

Pattern	Matches	Examples
beg.n	Any char	begin begun beg3n beg n
woodchucks?	Optional s	woodchuck woodchucks
to*	0 or more of previous char	t to too too
to+	1 or more of previous char	to too tooo toooo

- **Anchors ^ $**

Pattern	Matches
^[A-Z]	Palo Alto
^[^A-Za-z]	1 "Hello"
\.$	The end.
.$	The end? The end!

- **Substitutions:** Substitution in Python and UNIX commands: s/regexp1/pattern/
  e.g.: s/colour/color/

- **Words:** Eg: they lay back on the San Francisco grass and looked at the stars and their
  Type: an element of the vocabulary.
  Token: an instance of that type in running text.
  How many?
  - 15 tokens (or 14)
  - 13 types (or 12) (or 11?)

  N = number of tokens
  V = vocabulary = set of types, $|V|$ is size of vocabulary
  Heaps Law = Herdan's Law = $|V| = kN^{\beta}$
  where often $.67 < \beta < .75$
  i.e., vocabulary size grows with > square root of the number of word tokens

- **Corpora:** A text is produced by
  - a specific writer(s),
  - at a specific time,
  - in a specific variety,
  - of a specific language,
  - for a specific function.

➢ Text Normalization

1. Tokenizing (segmenting) words
2. Normalizing word formats
3. Segmenting sentences

1. Tokenizing (segmenting) words

- Space-based tokenization
- Word tokenization
- Subword Tokenization
  - Byte pair encoding (BPE)
  - Unigram modeling tokenization
  - Word piece

2. Normalizing word formats

- Putting words/tokens in a standard format
  - U.S.A. or USA
  - uhhuh or uh-huh
  - Fed or fed
  - am, is, be, are
- **Lemmatization**: Represent all words as their lemma, their shared root = dictionary headword form:
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*
- **Stemming**: Reduce terms to stems, chopping off affixes crudely

complete → complet

3. **Segmenting sentences**

- The most useful cues for segmenting a text into sentences are punctuation, like periods, question marks, and exclamation points.
- Question marks and exclamation points are relatively unambiguous markers of sentence boundaries.
- Periods are more ambiguous.
- The period character "." is ambiguous between a sentence boundary marker and a marker of abbreviations like Mr. or Inc.

➢ Edit Distance

- The minimum edit distance between two strings
- Is the minimum number of editing operations
  - Insertion
  - Deletion
  - Substitution
- Needed to transform one into the other

```
I N T E * N T I O N
|   |   |   |   |   |   |   |   |
* E X E C U T I O N
```

- If each operation has cost of 1
  - Distance between these is 5
- If substitutions cost 2 (Levenshtein)
  - Distance between them is 8

MORPHOLOGICAL ANALYSIS

- **Morphology** is the study of the internal structure of words. Morphology focuses on how the components within a word (stems, root words, prefixes, suffixes, etc.) are arranged or modified to create different meanings.
- English, for example, often ***adds "-s" or "-es" to the end of count nouns to indicate plurality, and a "-d" or "-ed" to a verb to indicate past tense. The suffix "-ly" is added to adjectives to create adverbs (for example, "happy" [adjective] and "happily" [adverb])***
- **Morphemes:** smallest part of words. For example,

"fox" consists of a **single morpheme (the morpheme fox)**
"cats" consist of two: **the morpheme cat and the morpheme -s**

- **Affixes are further divided into prefixes, suffixes, infixes, and circumfixes**. ***Prefixes*** precede the stem, ***suffixes*** follow the stem, ***circumfixes*** do both, and ***infixes*** are inserted inside the stem.

For example,
"**eats**" is composed of a stem **eat** and the suffix **-s.**
"**unbuckle**" is composed of a stem **buckle** and the prefix **un-.**

- Prefixes and suffixes are often called ***concatenative morphology*** since a word is composed of a number of morphemes concatenated together. A number of languages have extensive ***non-concatenative morphology*** (templatic morphology or root-and-pattern morphology), in which morphemes are combined in more complex ways. Another kind of ***non-concatenative morphology***.

➢ METHODS OF MORPHOLOGY

• **Morpheme Based Morphology**: In these words, are analysed as arrangements of morphemes. Word-based morphology is (usually) a ***word-and-paradigm approach***. Instead of stating rules to combine morphemes into word forms or to generate word forms from stems, word-based morphology states generalizations that hold between the forms of inflectional paradigms.
• **Lexeme Based Morphology:** Lexeme-based morphology usually takes what it is called an ***"item-and-process" approach***. Instead of analysing a word form as a set of morphemes arranged in sequence, a word form is said to be the result of applying rules that alter a word-form or steam in order to produce a new one.
• **Word based Morphology:** Word-based morphology is usually a ***word-and -paradigm approach*** instead of stating rules to combine morphemes into word forms.

➢ INFLECTIONAL, DERIVATIONAL

- **Inflectional morphology**
  - Adds information to a word consistent with its context wi
  - Examples
    - Number (singular versus plural)
      automaton → automata
    - *Walk → walks*
    - Case (nominative versus accusative versus...)
      he, him, his, ...
- **Derivational morphology**
  - Creates new words with new meanings (and often with ne
    speech)

➢ CLITICIZATION

In morphosyntax, **cliticization** is a process by which a complex word is formed by attaching a clitic to a fully inflected word.
Clitic: a morpheme that acts like a word but is reduced and attached to another word.
I've, l'opera

- **MORPHOLOGICAL ANALYSIS**

Morphological analysis:

token → lemma + part of speech + grammatical features

Examples:

cats → cat+N+plur
played → play+V+past
katternas → katt+N+plur+def+gen

TOKENIZATION

Tokenization is the process of dividing a text into **smaller units** known as ***tokens***. Tokens are typically **words or sub-words** in the context of natural language processing.

Types of Tokenization

1. **Word Tokenization**: Word tokenization divides the text into individual words. Example:
   Input: "Tokenization is an important NLP task."
   Output: ["Tokenization", "is", "an", "important", "NLP", "task", "."]
2. **Sentence Tokenization**: The text is segmented into sentences during sentence tokenization. Example:
   Input: "Tokenization is an important NLP task. It helps break down text into smaller units."
   Output: ["Tokenization is an important NLP task.", "It helps break down text into smaller units."]
3. **Subword Tokenization**: Subword tokenization entails breaking down words into smaller units, which can be especially useful when dealing with morphologically rich languages or rare words.
   Example:
   Input: "tokenization"
   Output: ["token", "ization"]
4. **Character Tokenization**: This process divides the text into individual characters.
   Example:
   Input: "Tokenization"
   Output: ["T", "o", "k", "e", "n", "i", "z", "a", "t", "i", "o", "n"]

Tokenization Use Cases

- **Search Engines**: Search engines use tokenization to process and understand user queries.
- **Machine Translation**: Tools like Google Translate rely on tokenization to convert sentences.
- **Speech Recognition**: Siri, Alexa use tokenization to process spoken language.

Limitations of tokenization

- **May struggle with ambiguity**: Tokenization with handling language ambiguities are difficult
- **Can be resource intensive**: Time-consuming, if need to define a specialized rule.
- **Might have token loss**: Some tokenization processes lose information.
- **May struggle with punctuation**: Apostrophes or dashes, can sometimes be tricky

STEMMING

Stemming is the ***process of reducing a word*** to its word stem that **affixes** to **suffixes** and prefixes or to the roots of words known as a **lemma**.

Types of Stemmer

1. Porter Stemmer: Martin Porter invented the Porter Stemmer or Porter algorithm in 1980. Five steps of word reduction are used in the method, each with its own set of mapping rules. Porter Stemmer is the original stemmer and is renowned for its ease of use and rapidity. Frequently, the resultant stem is a shorter word with the same root meaning.

Step 1a

SSES → SS
IES → I
SS → SS
S →

Step 1b

(m>0) EED → EE
(*v*) ED →
(*v*) ING →

If the second or third of the rules in Step 1b is successful, the following is performed.

AT → ATE
BL → BLE
IZ → IZE
(*d and not (*L or *S or *Z)) → single letter
(m=1 and *o) → E

Step 1c

(*v*) Y → I

Step 2

(m>0) ATIONAL → ATE
(m>0) TIONAL → TION
(m>0) ENCI → ENCE
(m>0) ANCI → ANCE
(m>0) IZER → IZE
(m>0) ABLI → ABLE
(m>0) ALLI → AL
(m>0) ENTLI → ENT
(m>0) ELI → E
(m>0) OUSLI → OUS
(m>0) IZATION → IZE
(m>0) ATION → ATE
(m>0) ATOR → ATE
(m>0) ALISM → AL
(m>0) IVENESS → IVE
(m>0) FULNESS → FUL
(m>0) OUSNESS → OUS
(m>0) ALITI → AL
(m>0) IVITI → IVE
(m>0) BILITI → BLE

Step 3

(m>0) ICATE → IC
(m>0) ATIVE →
(m>0) ALIZE → AL
(m>0) ICITI → IC
(m>0) ICAL → IC

(m>0) FUL →
(m>0) NESS →

Step 4
(m>1) AL →
(m>1) ANCE →
(m>1) ENCE →
(m>1) ER →
(m>1) IC →
(m>1) ABLE →
(m>1) IBLE →
(m>1) ANT →
(m>1) EMENT →
(m>1) MENT →
(m>1) ENT →
(m>1 and (*S or *T)) ION →
(m>1) OU →
(m>1) ISM →
(m>1) ATE →
(m>1) ITI →
(m>1) OUS →
(m>1) IVE →
(m>1) IZE →

Step 5a
(m>1) E →
(m=1 and not *o) E →

Step 5b
(m > 1 and *d and *L) → single letter

Example:

Connects ---> connect
Connecting ---> connect
Connections ---> connect
Connected ---> connect
Connection ---> connect
Connectings ---> connect
Connect ---> connect

2. Snowball Stemmer: Martin Porter also created Snowball Stemmer. The method utilized in this instance is more precise and is referred to as *"English Stemmer" or "Porter2 Stemmer."* It is somewhat faster and more logical than the original Porter Stemmer.

Few Rules:

ILY -----> ILI
LY -----> Nil
SS -----> SS
S -----> Nil
ED -----> E,Nil

Example:

generous ---> generous
generate ---> generat
generously ---> generous
generation ---> generat

3. Lancaster Stemmer: Lancaster Stemmer is straightforward, although it often produces results with excessive stemming. Over-stemming renders stems non-linguistic or meaningless. *Applies rules to strip suffixes such as "ing" or "ed."*

eating ---> eat
eats ---> eat
eaten ---> eat
puts ---> put
putting ---> put

4. Regexp Stemmer: Regex stemmer identifies morphological affixes using regular expressions. *Substrings matching the regular expressions will be discarded*.

mass ---> mas
was ---> was
bee ---> bee
computer ---> computer
advisable ---> advis

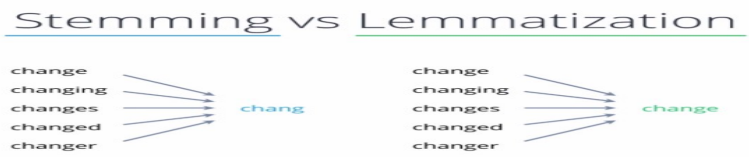
LEMMATIZATION

Lemmatisation (or lemmatization) in linguistics is the ***process of grouping together the inflected forms of a word so they can be analysed as a single item***, identified by the word's lemma, or dictionary form. The association of the base form with a part of speech is often called a **lexeme** of the word.

For example: the verb 'to ***walk'*** may appear as ***'walk'***, ***'walked'***, ***'walks'*** or ***'walking'***. The base form, 'walk', that one might look up in a **dictionary**, is called the **lemma** for the word.

For instance:

- The word "**better**" has "**good**" as its lemma. This link is missed by stemming, as it requires a dictionary look-up.
- The word "**walk**" is the base form for the word "**walking**", and hence this is matched in both stemming and lemmatisation.
- The word "**meeting**" can be either the base form of a noun or a form of a verb ("**to meet**") depending on the context; e.g., "**in our last meeting**" or "**We are meeting again tomorrow**". Unlike stemming, lemmatisation attempts to select the correct lemma depending on the context.



Lemmatization Techniques

1. Rule Based Lemmatization: Rule-based lemmatization involves the application of predefined rules to derive the base or root form of a word.

Rule: For regular verbs ending in “-ed,” remove the “-ed” suffix.

Example: “walked” --> "walk"

2. Dictionary-Based Lemmatization: Dictionary-based lemmatization relies on predefined dictionaries or lookup tables to map words to their corresponding base forms or lemmas.

‘running’ -> ‘run’

‘better’ -> ‘good’

‘went’ -> ‘go’

3. Machine Learning-Based Lemmatization: Machine learning-based lemmatization leverages computational models to automatically learn the relationships between words and their base forms. Example: When encountering the word 'went,' the model, having learned patterns, predicts the base form as 'go.'

FEATURE EXTRACTION

- Feature extraction is a process in machine learning and data analysis that ***involves identifying and extracting relevant features from raw dat***a.
- ***NLP feature extraction methods are techniques used to convert raw text data into numerical representations that can be processed by machine learning models.***
- These methods aim to capture the meaningful information and patterns in text data.

1. Label Encoding
2. One Hot Encoding
3. Count Vectorization
   TF-IDF Vectorizer
   Bag of Words (BOW)
4. Word Embedding
   Word2Vec
   GloVe
   FastText
5. N-gram Features

TF-IDF stands for **term frequency-inverse document frequency**. ***The TF–IFD value increases proportionally to the number of times a word appears in the document and decreases with the number of documents in the corpus that contain the word.*** It is composed of 2 sub-parts, which are:

1. **Term Frequency (TF)**
2. **Inverse Document Frequency (IDF**)

TF-IDF is used for:

1. Text retrieval and information retrieval systems
2. Document classification and text categorization
3. Text summarization
4. Feature extraction for text data in machine learning algorithms.

1. **Term Frequency (TF):** TF measures the frequency of a term within a document. It is calculated as the ratio of the number of times a term occurs in a document to the total number of terms in that document. *The goal is to emphasize words that are frequent within a document.*

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

2. **Inverse Document Frequency (IDF):** IDF measures the rarity of a term across a collection of documents. It is calculated as the logarithm of the ratio of the total number of documents to the number of documents containing the term. *The goal is to penalize words that are common across all documents.*

$$IDF(t,D) = \log\left(\frac{\text{Total number of documents in the corpus } N}{\text{Number of documents containing term } t}\right)$$

3. **Document Frequency**: This tests the meaning of the text, which is very similar to TF, in the whole corpus collection. The only difference is that in document d, TF is the frequency counter for a term t, while df is the number of occurrences in the document set N of the term t. In other words, the number of papers in which the word is present is DF.

   df(t) = occurrence of t in documents

Combining TF and IDF: The TF-IDF score for a term in a document is obtained by multiplying its TF and IDF scores.

$$\text{TF-IDF}(t,d,D)=TF(t,d)\times IDF(t,D)$$

Modeling using TF-IDF

Imagine the term t appears 20 times in a document that contains a total of 100 words. Term Frequency (TF) of t can be calculated as follow:

TF=20/100=0.2

Assume a collection of related documents contains 10,000 documents. If 100 documents out of 10,000 documents contain the term t, Inverse Document Frequency (IDF) of t can be calculated as follows

IDF=log (10000/100) = 2

Using these two quantities, we can calculate TF-IDF score of the term t for the document.

TF-IDF=0.2∗2=0.4

PARTS OF SPEECH TAGGING

Part-of-speech tagging is the process of assigning ***a part-of-speech to each word in a text***.

POS Tagging (Parts of Speech Tagging) is a process to mark up the words in text format for a particular part of a speech based on its definition and context. It is responsible for text reading in a language and assigning some specific token (Parts of Speech) to each word. It is also called ***grammatical tagging.***

Tagging is a disambiguation task; words are ambiguous—have more than one possible part-of-speech—and the goal is to find the correct tag for the situation.

For example,

"**book"** can be a **verb** ("*book that flight"*) or a **noun** ("*hand me that book"*).

"**That**" can be a **determiner** ("*Does that flight serve dinner"*) or a **complementizer** ("*I thought that your flight was earlier"*).

The input is a sequence $x_1, x_2, \ldots x_n$ of (tokenized) words and a tagset, and the output is a sequence $y_1, y_2, \ldots y_n$ of tags, each output y_i corresponding exactly to one input x_i,

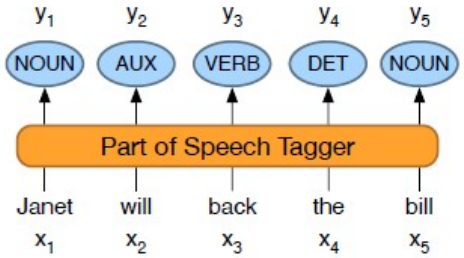


Figure: The task of part-of-speech tagging: mapping from input words $x_1, x_2, \ldots x_n$ to output POS tags $y_1, y_2, \ldots y_n$.

Input: Everything to permit us.

Output: [('Everything', NN),('to', TO), ('permit', VB), ('us', PRP)]

NLTK POS Tags Examples are as below:

Abbreviation	Meaning
CC	coordinating conjunction
CD	cardinal digit
DT	Determiner
EX	existential there
FW	foreign word
IN	preposition/subordinating conjunction

JJ	This NLTK POS Tag is an adjective (large)
JJR	adjective, comparative (larger)
JJS	adjective, superlative (largest)
LS	list marker
MD	modal (could, will)
NN	noun, singular (cat, tree)
NNS	noun plural (desks)
NNP	proper noun, singular (sarah)
NNPS	proper noun, plural (indians or americans)
PDT	predeterminer (all, both, half)
POS	possessive ending (parent\ ‘s)
PRP	personal pronoun (hers, herself, him, himself)
PRP$	possessive pronoun (her, his, mine, my, our)
RB	adverb (occasionally, swiftly)
RBR	adverb, comparative (greater)
RBS	adverb, superlative (biggest)
RP	particle (about)
TO	infinite marker (to)
UH	interjection (goodbye)
VB	verb (ask)
VBG	verb gerund (judging)
VBD	verb past tense (pleaded)
VBN	verb past participle (reunified)
VBP	verb, present tense not 3rd person singular(wrap)
VBZ	verb, present tense with 3rd person singular (bases)
WDT	wh-determiner (that, what)
WP	wh- pronoun (who)
WRB	wh- adverb (how)

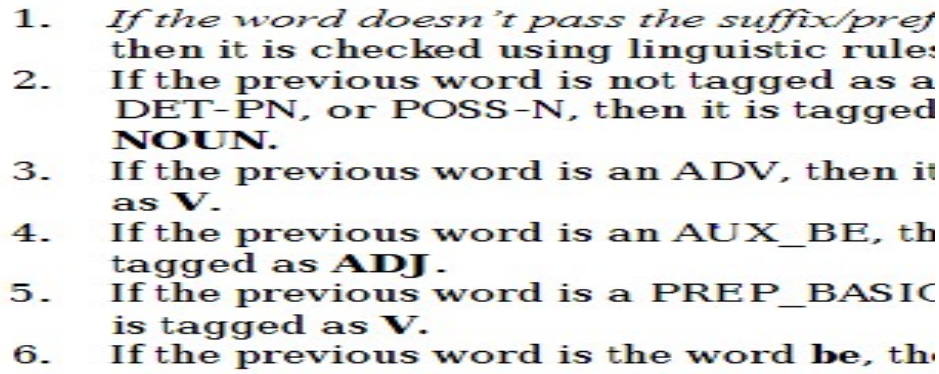
Two classes of words: Open vs. Closed

- **Closed class words**
  - Relatively fixed membership
  - Usually **function** words: short, frequent words with grammatical function
    - determiners: ***a, an, the***
    - pronouns: ***she, he, I***
    - prepositions: ***on, under, over, near, by, …***
- **Open class words**
  - Usually **content** words: Nouns, Verbs, Adjectives, Adverbs
    - Plus interjections: **oh, ouch, uh-huh, yes, hello**
  - New nouns and verbs like *iPhone* or *to fax*

	Tag	Description	Example
Open Class	**ADJ**	Adjective: noun modifiers describing properties	*red*, *young*, *awesome*
	ADV	Adverb: verb modifiers of time, place, manner	*very*, *slowly*, *home*, *yesterday*
	NOUN	words for persons, places, things, etc.	*algorithm*, *cat*, *mango*, *beauty*
	VERB	words for actions and processes	*draw*, *provide*, *go*
	PROPN	Proper noun: name of a person, organization, place, etc..	*Regina*, *IBM*, *Colorado*
	INTJ	Interjection: exclamation, greeting, yes/no response, etc.	*oh*, *um*, *yes*, *hello*
Closed Class Words	**ADP**	Adposition (Preposition/Postposition): marks a noun's spacial, temporal, or other relation	*in, on, by under*
	AUX	Auxiliary: helping verb marking tense, aspect, mood, etc.,	*can, may, should, are*
	CCONJ	Coordinating Conjunction: joins two phrases/clauses	*and*, *or*, *but*
	DET	Determiner: marks noun phrase properties	*a, an, the, this*
	NUM	Numeral	*one, two, first, second*
	PART	Particle: a preposition-like form used together with a verb	*up, down, on, off, in, out, at, by*
	PRON	Pronoun: a shorthand for referring to an entity or event	*she, who, I, others*
	SCONJ	Subordinating Conjunction: joins a main clause with a subordinate clause such as a sentential complement	*that*, *which*
Other	**PUNCT**	Punctuation	; , ()
	SYM	Symbols like $ or emoji	$, %
	X	Other	asdf, qwfg

Techniques for POS tagging

1. **Rule-based POS tagging**: The rule-based POS tagging models apply a set of handwritten rules and use contextual information to assign POS tags to words. These rules are often known as context frame rules. *One such rule might be: "If an ambiguous/unknown word ends with the suffix 'ing' and is preceded by a Verb, label it as a Verb".*



2. **Transformation Based Tagging**: The transformation-based approaches use a pre-defined set of handcrafted rules as well as automatically induced rules that are generated during training.
3. **Deep learning models**: Various Deep learning models have been used for POS tagging such as Meta-BiLSTM which have shown an impressive accuracy of around 97 percent.
4. **Stochastic (Probabilistic) tagging**: A stochastic approach *includes frequency, probability or statistics.* The simplest stochastic approach finds out the most frequently used tag for a specific word in the annotated training data and uses this information to tag that word in the unannotated text. But sometimes this approach comes up with sequences of tags for sentences that are not acceptable according to the grammar rules of a language. *One such approach is to calculate the probabilities of various tag sequences that are possible for a sentence and assign the POS tags from the sequence with the highest probability. Hidden Markov Models (HMMs) are probabilistic approaches to assign a POS Tag.*

NAMED ENTITY RECOGNITION

- It highlights the fundamental **concepts and references** in the text. ***NER identifies entities such as people, locations, organizations, dates, etc. from the text***. NER is generally based on grammar rules and supervised models.
- A **named entity** is, roughly speaking, anything that can be referred to with a proper name: a person, a location, an organization.
- The task of **named entity recognition (NER)** is to find spans of text that constitute proper names and tag the type of the entity. Four entity tags are most common: **PER (person), LOC (location), ORG (organization), or GPE (geo-political entity).**
- However, the term named entity is commonly extended to include things **that aren't entities per se, including dates, times, and other kinds of temporal expressions, and even numerical expressions like prices.** Here's an example of the output of an NER tagger:

Citing high fuel prices, [ORG **United Airlines**] said [TIME **F**
has increased fares by [MONEY **$6**] per round trip on flights
cities also served by lower-cost carriers. [ORG **American Ai**
unit of [ORG **AMR Corp.**], immediately matched the move, sp
[PER **Tim Wagner**] said. [ORG **United**], a unit of [ORG **UAI**
said the increase took effect [TIME **Thursday**] and applies
routes where it competes against discount carriers, such as [LO
to [LOC **Dallas**] and [LOC **Denver**] to [LOC **San Francisco**].

The text contains 13 mentions of named entities including 5 organi
tions, 2 times, 1 person, and 1 mention of money. Figure 8.5 shows

Type	Tag	Sample Categories	Example sentences
People	PER	people, characters	**Turing** is a giant of cor
Organization	ORG	companies, sports teams	The **IPCC** warned abou
Location	LOC	regions, mountains, seas	**Mt. Sanitas** is in **Suns**
Geo-Political Entity	GPE	countries, states	**Palo Alto** is raising the

➢ Ambiguity in NE

- For a person, the category definition is intuitively quite clear, but for computers, there is some ambiguity in classification. Let's look at some ambiguous example:
  - ***England** (Organisation)* won the 2019 world cup vs The 2019 world cup happened in ***England** (Location)*.
  - ***Washington** (Location)* is the capital of the US vs The first president of the US was ***Washington** (Person)*.

➢ Methods of NER

- One way is to train the model for ***multi-class classification*** using different machine learning algorithms, but it requires a lot of labelling.
- In addition to ***labelling the model*** also requires a deep understanding of context to deal with the ambiguity of the sentences. This makes it a challenging task for simple machine learning /
- Another way is that ***Conditional random field*** that is implemented by both NLP Speech Tagger and NLTK. It is a probabilistic model that can be used to model sequential data such as words. The CRF can capture a deep understanding of the context of the sentence. In this model, the input

$$\mathbf{X} = \{\vec{x}_1, \vec{x}_2, \vec{x}_3, \ldots, \vec{x}_T\}$$

$$\mathrm{p}(\mathrm{y}|\mathbf{x}) = \frac{1}{z(\vec{x})} \prod^{T} \exp\left\{ \sum^{k} \omega_k f_k (y_t,$$

- ***Deep Learning Based NER:*** Deep Learning NER is much more accurate than previous method, as it is capable to assemble words. This is due to the fact that it used a method

called ***word embedding***, that is capable of understanding the semantic and syntactic relationship between various words.

- ***Rule Based Named Entity Recognition:*** Rules are there:
  1. **If words are repeated, return them as the name.**
  2. **If words are between quotation marks, return them as the name.**
  3. **Return words before company suffixes such as "Inc" as the name**
  4. **Return the entire memo as name if the word count in the memo is fewer than 3. If none of the extraction patterns is successful, the program outputs nothing and lets the user know that the program has failed to extract a vendor name from the memo.**

N-GRAMS

An N-gram means a sequence of N words.
For example,
"**Medium blog**" is a 2-gram (a **bigram**)
"**A Medium blog post**" is a **4-gram**,
"**Write on Medium**" is a 3-gram (**trigram**).

The following sentences as the training corpus:

- Thank you so much for your help.
- I really appreciate your help.
- Excuse me, do you know what time it is?
- I'm really sorry for not inviting you.
- I really like your watch.

Suppose we're calculating the probability of word "w1" occurring after the word "w2," then the formula for this is as follows:

count(w2 w1) / count(w2)

which is the number of times the words occurs in the required sequence, divided by the number of the times the word before the expected word occurs in the corpus.
From our example sentences, let's calculate the probability of the word "like" occurring after the word "really":
count(really like) / count(really)
= 1 / 3
= 0.33

Similarly, for the other two possibilities:
count(really appreciate) / count(really)
= 1 / 3
= 0.33
count(really sorry) / count(really)
= 1 / 3
= 0.33

Language Model with N-Gram

N	Term
1	Unigram
2	Bigram
3	Trigram
N	n-gram

Now how can we compute probabilities of entire sequences like $P(w_1$
One thing we can do is decompose this probability using the **chain rul**
bility:

$$P(X_1 \ldots X_n) = P(X_1)P(X_2|X_1)P(X_3|X_{1:2}) \ldots P(X_n|X_{1:n-1})$$
$$= \prod_{k=1}^{n} P(X_k|X_{1:k-1})$$

Applying the chain rule to words, we get

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2}) \ldots P(w_n|w_{1:n-1})$$

The **bigram** model, for example, approximates the probability
all the previous words $P(w_n|w_{1:n-1})$ by using only the conditional p
preceding word $P(w_n|w_{n-1})$. In other words, instead of computing

$$P(\text{the}|\text{Walden Pond's water is so transparent that})$$

we approximate it with the probability

$$P(\text{the}|\text{that})$$

When we use a bigram model to predict the conditional probability
we are thus making the following approximation:

How do we estimate these bigram or n-gram probabilities? An
estimate probabilities is called **maximum likelihood estimation** o
the MLE estimate for the parameters of an n-gram model by gettin
corpus, and **normalizing** the counts so that they lie between 0 and 1

For example, to compute a particular bigram probability of a
previous word x, we'll compute the count of the bigram $C(xy)$ and n
sum of all the bigrams that share the same first word x:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum C(w_{n-1}w)}$$

Applications of n-grams:

spelling error detection and correction, query expansion, information retrieval with serial, inverted and signature files, dictionary look-up, text compression, and language identification.

VARIETY OF SMOOTHING

To keep the model from assigning zero probability to unseen words/n-grams, we have to shave off a bit of probability mass from some more frequent words/n-grams and give it to those who have never seen.

Smoothing is the task of re-evaluating some of the zero probability and low probability estimations.

1. Laplace (add-one) smoothing
2. Add-k smoothing
3. Good Turing smoothing
4. Stupid backoff and Interpolation
5. Kneser-Ney smoothing.

1. LAPLACE SMOOTHING (ADD-ONE)

Example: "I reside in Bengaluru"

SL.No.	Type of n-gram	Generated n-grams
1	Unigram	["I","reside","in", "Bengaluru"]
2	Bigram	["I reside","reside in","in Bengaluru"]
3	Trigram	["I reside in", "reside in Bengaluru"]

For Example:

- Let's take an example of the sentence: ***'Natural Language Processing'.*** For predicting the first word, let's say the word has the following probabilities:

word	P(word | <s
The	0.4
Processing	0.3
Natural	0.12

- Now, we know the probability of getting the first word as natural. But, what's the probability of getting the next word after getting the word '*Language*' after the word '*Natural*'.

word	P(word | 'Nat
The	0.05
Processing	0.3
Natural	0.15

- After getting the probability of generating words 'Natural Language', what's the probability of getting '*Processing*'.

word	P(word | 'Language')
The	0.1
Processing	0.7
Natural	0.1

Probablity:

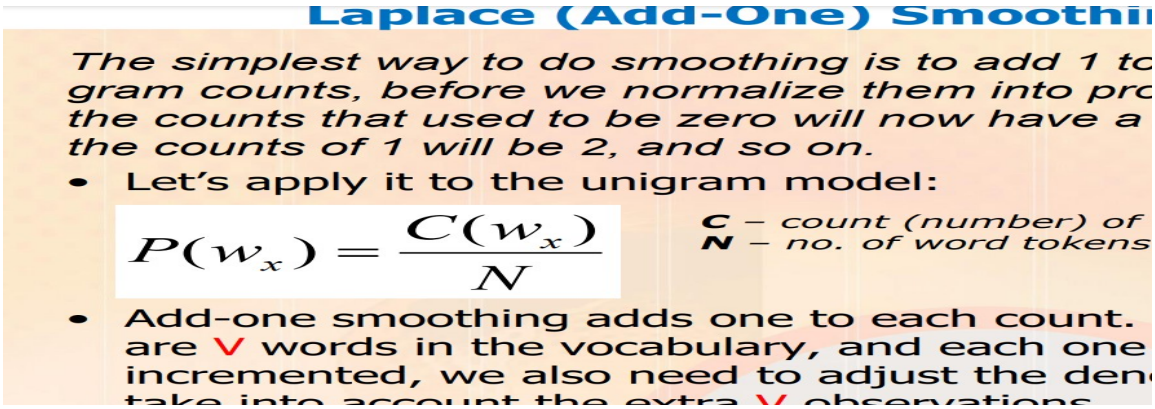
P(w|h), the probability of a word w given some history h. Suppose the history h is "its water is so transparent that" and we want to know the probability that the next word is the:

$$P(the|its\ water\ is\ so\ transparen$$

One way to estimate this probability is from relative frequency counts: take a very large corpus, count the number of times we see its water is so transparent that, and count the number of times this is followed by the. ***This would be answering the question "Out of the times we saw the history h, how many times was it followed by the word w",*** as follows:

$$P(the|its\ water\ is\ so\ transparent\ t$$
$$\frac{C(its\ water\ is\ so\ transparent\ th}{C(its\ water\ is\ so\ transparent}$$

To estimate the probability of a word w given a history h, or the probability of an entire word sequence.



2. Add-k smoothing

One alternative to add-one smoothing is to move a bit less of the probability mass from the seen to the unseen events. Instead of adding 1 to each count, we add a fractional count k. This algorithm is therefore called ***add-k smoothing***.

$$P^*_{\text{Add-k}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w}{C(w_{n-1})}$$

Although add-k is useful for some tasks (including text classification), it turns out that it still doesn't work well for language modeling, generating counts with poor variances and often inappropriate discounts.

3. Good Turing Smoothing

Re-estimate the amount of probability mass to assign to N-grams with zero or low counts by looking at the number of N-grams with higher counts.
Probabilities based on the number of times events of different frequencies occur, assigning higher probabilities to unseen events.
Good-Turing estimate gives a smoothed count c^* based on the set Nc for all c, as follows:

$$C^* = (C+1)\frac{N_C + 1}{N_C}$$

4. Stupid backoff and Interpolation

- **Backoff**: If a higher-order n-gram has zero counts, the model "backs off" to a lower-order n-gram (e.g., from trigrams to bigrams).
- **Interpolation**: Combines probabilities from different n-gram levels using weights.

$$S(w_i|w_{i-N+1:i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-N+1:i})}{\text{count}(w_{i-N+1:i-1})} & \text{if count}(w_{i-N+1:i}) > 0 \\ \lambda S(w_i|w_{i-N+2:i-1}) & \text{otherwise} \end{cases}$$

5. Kneser-Ney smoothing

Redistributes probabilities to less frequent n-grams while maintaining the distribution of high-frequency n-grams.

$$P(w_i|w_{i-1}) = \max(C(w_{i-1}, w_i) - D, 0)/C(w_{i-1}) + \lambda(w_{i-1}) \cdot P_{continuation}(w_i)$$

where:

- D is a discount parameter,
- $\lambda(w_{i-1})$ is a normalization factor.