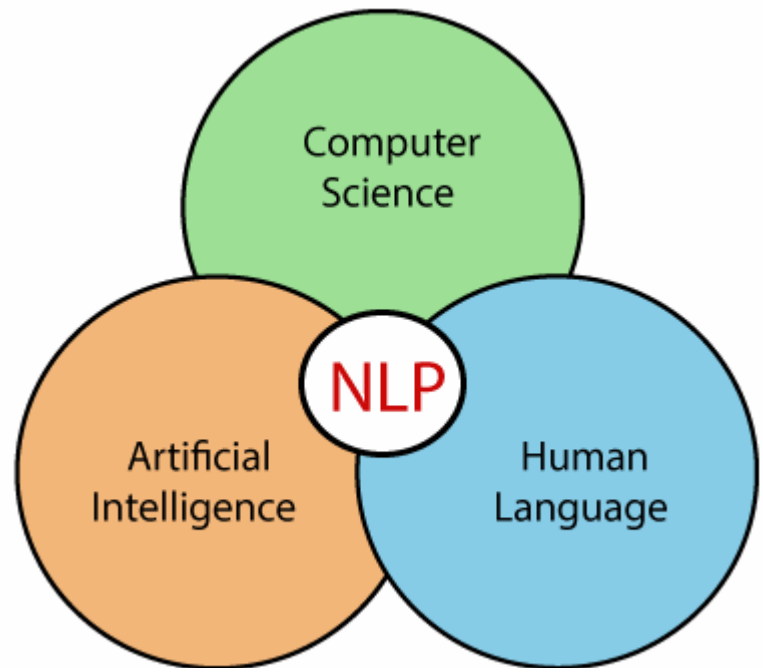
Natural Language Processing (NLP) Unit-I

Introduction to Natural Lanaguage 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 Recognization, ,N-grams , Smoothing

NATURAL LANGUAGE PROCESSING – INTRODUCTION

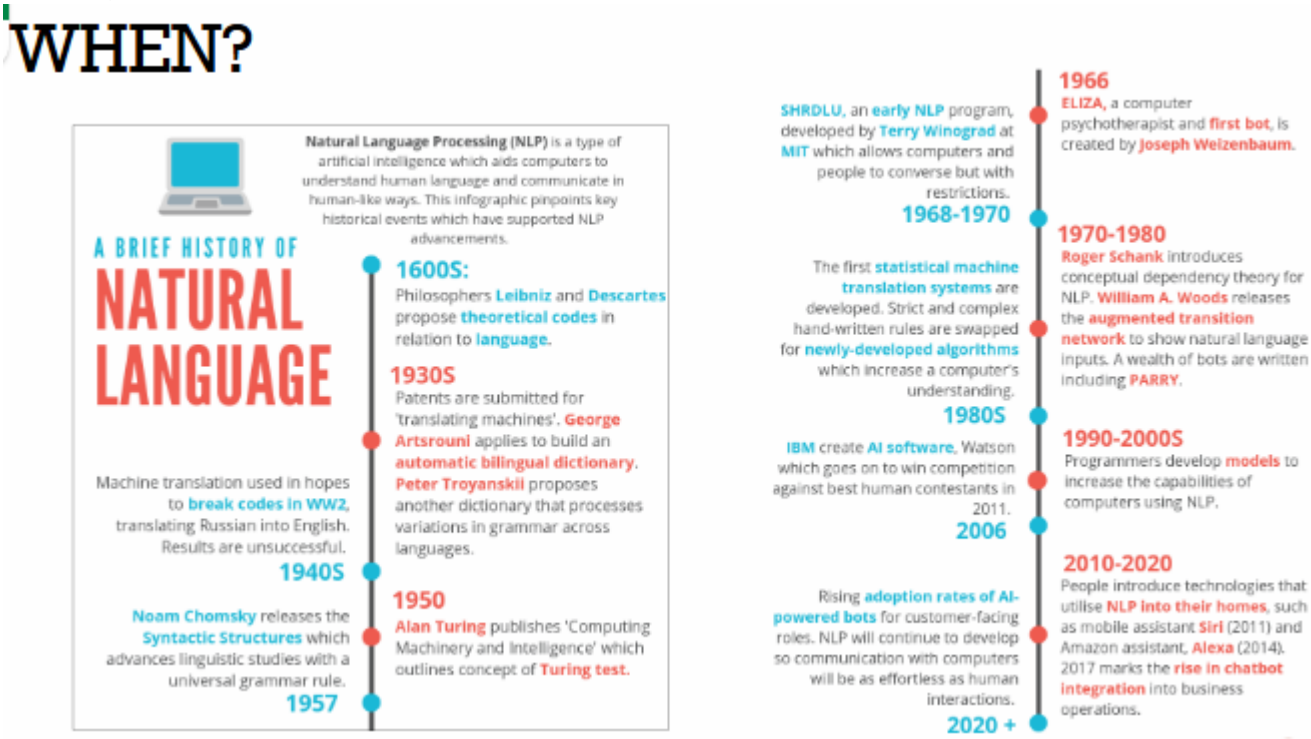
- Humans communicate through some form of language either by text or speech.
- To make interactions between computers and humans, computers need to understand natural languages used by humans.
- Natural language processing is all about making computers learn, understand, analyze, manipulate and interpret natural(human) languages.
- NLP stands for Natural Language Processing, which is a part of Computer Science, Human languages or Linguistics, and Artificial Intelligence.
- Processing of Natural Language is required when you want an intelligent system like robot to perform as per your instructions, when you want to hear decision from a dialogue based clinical expert system, etc.
- The ability of machines to interpret human language is now at the core of many applications that we use every day - chatbots, Email classification and spam filters, search engines, grammar checkers, voice assistants, and social language translators.
- The input and output of an NLP system can be Speech or Written Text.



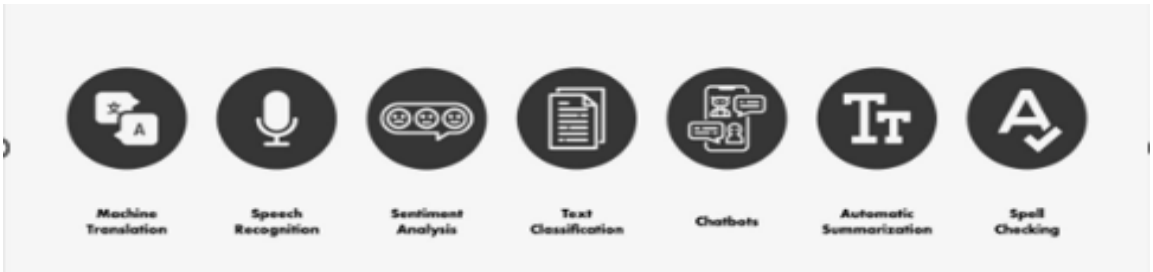
Defn : Natural Language Processing (NLP) is a sub-field of Artificial Intelligence that is focused on enabling computers to understand and

process human languages, to get computers closer to a human-level understanding of language.

History



APPLICATIONS of NLP or Use cases of NLP



1. Sentiment analysis

- **Sentiment analysis**, also referred to as **opinion mining**, is an approach to natural language processing (NLP) that identifies the emotional tone behind a body of text.
- This is a popular way for organizations to determine and categorize opinions about a product, service or idea.

- Sentiment analysis systems help organizations gather insights into real-time customer sentiment, customer experience and brand reputation.
- Generally, these tools use text analytics to analyze online sources such as emails, blog posts, online reviews, news articles, survey responses, case studies, web chats, tweets, forums and comments.
- Sentiment analysis uses machine learning models to perform text analysis of human language. The metrics used are designed to detect whether the overall sentiment of a piece of text is positive, negative or neutral.

2. Machine Translation

- **Machine translation**, sometimes referred to by the abbreviation **MT**, is a sub-field of computational linguistics that investigates the use of software to translate text or speech from one language to another.
- On a basic level, MT performs mechanical substitution of words in one language for words in another, but that alone rarely produces a good translation because recognition of whole phrases and their closest counterparts in the target language is needed.
- Not all words in one language have equivalent words in another language, and many words have more than one meaning.
- Solving this problem with corpus statistical and neural techniques is a rapidly growing field that is leading to better translations, handling differences in linguistic typology, translation of idioms, and the isolation of anomalies.
- **Corpus:** A collection of written texts, especially the entire works of a particular author.

3. Text Extraction

- There are a number of natural language processing techniques that can be used to extract information from text or unstructured data.
- These techniques can be used to extract information such as entity names, locations, quantities, and more.
- With the help of natural language processing, computers can make sense of the vast amount of unstructured text data that is generated every day, and humans can reap the benefits of having this information readily available.
- Industries such as healthcare, finance, and e-commerce are already using natural language processing techniques to extract information and improve business processes.
- As the machine learning technology continues to develop, we will only see more and more information extraction use cases covered.

4. Text Classification

- Unstructured text is everywhere, such as emails, chat conversations, websites, and social media. Nevertheless, it's hard to extract value from this data unless it's organized in a certain way.
- Text classification also known as *text tagging* or *text categorization* is the process of categorizing text into organized groups. By using Natural Language Processing (NLP), text classifiers can automatically analyze text and then assign a set of pre-defined tags or categories based on its content.
- Text classification is becoming an increasingly important part of businesses as it allows to easily get insights from data and automate business processes.

5. Speech Recognition

- Speech recognition is an interdisciplinary subfield of computer science and computational linguistics that develops methodologies and technologies that enable the recognition and translation of spoken language into text by computers.
- It is also known as automatic speech recognition (ASR), computer speech recognition or speech to text (STT).
- It incorporates knowledge and research in the computer science, linguistics and computer engineering fields. The reverse process is speech synthesis.

Speech recognition use cases

- A wide number of industries are utilizing different applications of speech technology today, helping businesses and consumers save time and even lives. Some examples include:
- Automotive: Speech recognizers improves driver safety by enabling voice-activated navigation systems and search capabilities in car radios.
- Technology: Virtual agents are increasingly becoming integrated within our daily lives, particularly on our mobile devices. We use voice commands to access them through our smartphones, such as through Google Assistant or Apple's Siri, for tasks, such as voice search, or through our speakers, via Amazon's Alexa or Microsoft's Cortana, to play music. They'll only continue to integrate into the everyday products that we use, fueling the "Internet of Things" movement.
- Healthcare: Doctors and nurses leverage dictation applications to capture and log patient diagnoses and treatment notes.
- Sales: Speech recognition technology has a couple of applications in sales. It can help a call center transcribe thousands of phone calls between customers and agents to identify common call patterns and issues. AI chatbots can also talk to people via a webpage, answering common queries and solving basic requests without needing to wait for a contact center agent to be available. In both instances speech recognition systems help reduce time to resolution for consumer issues.

6. Chatbot

- Chatbots are computer programs that conduct automatic conversations with people. They are mainly used in customer service for information acquisition. As the name implies, these are bots designed with the purpose of chatting and are also simply referred to as "bots."
- You'll come across chatbots on business websites or messengers that give pre-scripted replies to your questions. As the entire process is automated, bots can provide quick assistance 24/7 without human intervention.

7. Email Filter

- One of the most fundamental and essential applications of NLP online is email filtering. It began with spam filters, which identified specific words or phrases that indicate a spam message. But, like early NLP adaptations, filtering has been improved.

- Gmail's email categorization is one of the more common, newer implementations of NLP. Based on the contents of emails, the algorithm determines whether they belong in one of three categories (main, social, or promotional).
- This maintains your inbox manageable for all Gmail users, with critical, relevant emails you want to see and reply to fast.

8. Search Autocorrect and Autocomplete

- When you type 2-3 letters into Google to search for anything, it displays a list of probable search keywords. Alternatively, if you search for anything with mistakes, it corrects them for you while still returning relevant results. Isn't it incredible?
- Everyone uses Google search autocorrect autocomplete on a regular basis but seldom gives it any thought. It's a fantastic illustration of how natural language processing is touching millions of people across the world, including you and me.
- Both, search autocomplete and autocorrect make it much easier to locate accurate results

Components of NLP

There are the following two components of NLP –

1. Natural Language Understanding (NLU)
   Natural Language Understanding (NLU) helps the machine to understand and analyse human language by extracting the metadata from content such as concepts, entities, keywords, emotion, relations, and semantic roles.
   NLU mainly used in Business applications to understand the customer's problem in both spoken and written language.
   NLU involves the following tasks -
   - It is used to map the given input into useful representation.
   - It is used to analyze different aspects of the language.

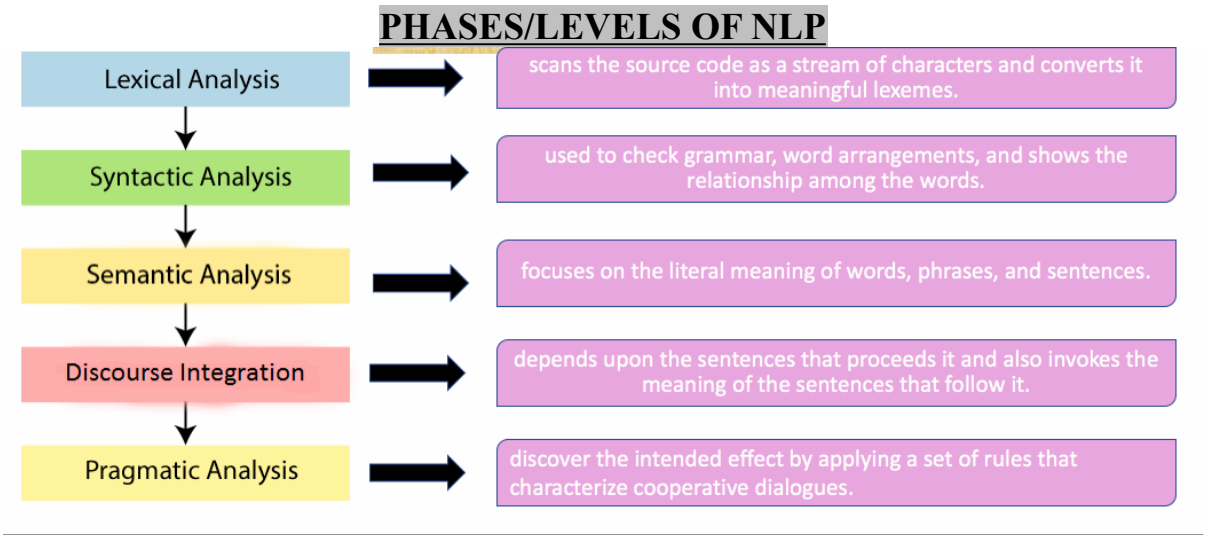
2**. Natural Language Generation (NLG)**
   Natural Language Generation (NLG) acts as a translator that converts the computerized data into natural language representation. It mainly involves Text planning, Sentence planning, and Text Realization.

Difference between NLU and NLG

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

NLP terminalogy

•Phonology − It is study of organizing sound systematically.
•Morphology − It is a study of construction of words from primitive meaningful units.
•Morpheme − It is primitive unit of meaning in a language.
•Syntax − It refers to arranging words to make a sentence. It also involves determining the structural role of words in the sentence and in phrases.
•Semantics − It is concerned with the meaning of words and how to combine words into meaningful phrases and sentences.
•Pragmatics − It deals with using and understanding sentences in different situations and how the interpretation of the sentence is affected.
•Discourse − It deals with how the immediately preceding sentence can affect the interpretation of the next sentence.
•World Knowledge − It includes the general knowledge about the world.



Lexical Analysis:

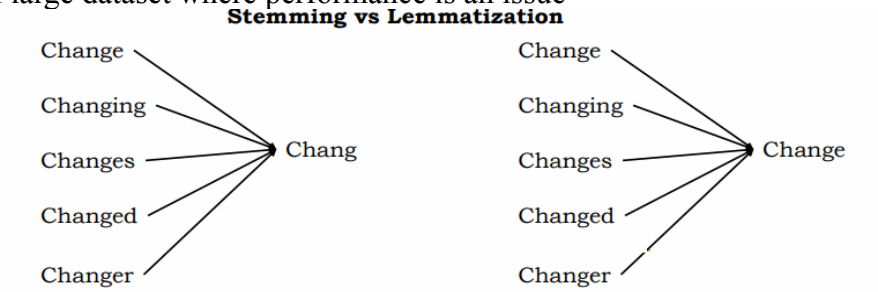
- The first phase of NLP is the Lexical Analysis.
- This phase scans the source code as a stream of characters and converts it into meaningful lexemes.
- It divides the whole text into paragraphs, sentences, and words.
- **Lexeme**: A lexeme is a basic unit of meaning. In linguistics, the abstract unit of morphological analysis that corresponds to a set of forms taken by a single word is called lexeme.
- The way in which a lexeme is used in a sentence is determined by its grammatical category.
- Lexeme can be individual word or multiword.
- For example, the word talk is an example of an individual word lexeme, which may have many grammatical variants like talks, talked and talking.
- Multiword lexeme can be made up of more than one orthographic word. For example, speak up, pull through, etc. are the examples of multiword lexemes.
- The most common lexicon normalization practices are:

• Stemming: Stemming is a rudimentary rule-based process of stripping the suffixes (“ing”, “ly”, “es”, “s” etc.) from a word.

• Lemmatization: Lemmatization, on the other hand, is an organized & step by step procedure of obtaining the root form of the word, it makes use of vocabulary (dictionary importance of words) and morphological analysis (word structure and grammar relations)

Lemmatization and Stemming, both are used to generate root form of derived (inflected) words. However, lemma is an actual language word, whereas stem may not be an actual word.

• For instance, stemming the word ‘Caring‘ would return ‘Car‘ and lemmatizing the word ‘Caring‘ would return ‘Care‘. Stemming is used in case of large dataset where performance is an issue



Syntax Analysis (Parsing)

- Syntactic Analysis is used to check grammar, word arrangements, and shows the relationship among the words.
- The syntactic analysis basically assigns a semantic structure to text. It is also known as syntax analysis or parsing. The word ‘parsing’ is originated from the Latin word ‘pars’ which means ‘part’. The syntactic analysis deals with the syntax of Natural Language. In syntactic analysis, grammar rules have been used.
- • For example: Correct Syntax: Sun rises in the east. Incorrect Syntax: Rise in sun the east.
- The sentence such as “The school goes to boy” is rejected by English syntactic analyzer.

Semantic Analysis

- Semantic analysis is concerned with the meaning representation.
- Semantic Analysis of Natural Language captures the meaning of the given text while taking into account context, logical structuring of sentences, and grammar roles.
- Semantic Analysis of Natural Language can be classified into two broad parts:

  • Lexical Semantic Analysis: Lexical Semantic Analysis involves understanding the meaning of each word of the text individually. It basically refers to fetching the dictionary meaning that a word in the text is deputed to carry.

  • Compositional Semantics Analysis: Although knowing the meaning of each word of the text is essential, it is not sufficient to completely understand the meaning of the text.

Consider the sentence: “The apple ate a banana”. Although the sentence is syntactically correct, it doesn’t make sense because apples can’t eat. The semantic analysis looks for meaning in the given sentence. It also deals with combining words into phrases.

- The semantic analyzer disregards sentence such as “hot ice-cream”.
- Another Example is “Manhattan calls out to Dave” passes a syntactic analysis because it’s a grammatically correct sentence. However, it fails a semantic analysis. Because Manhattan is a place (and can’t literally call out to people), the sentence’s meaning doesn’t make sense.

Discourse Integration

- Discourse Integration depends upon the sentences that precedes it and also invokes the meaning of the sentences that follow it.
- In the text, “Jack is a bright student. He spends most of the time in the library.” Here, discourse assigns “he” to refer to “Jack”.
- Another example, if one sentence reads, “Manhattan speaks to all its people,” and the following sentence reads, “It calls out to Dave,” discourse integration checks the first sentence for context to understand that “It” in the latter sentence refers to Manhattan.

Pragmatic Analysis

- During this, what was said is re-interpreted on what it actually meant.
- It involves deriving those aspects of language which require real world knowledge.

By analyzing the contextual dimension of the documents and queries, a more detailed representation is derived.

• Given a sentence, “Turn off the lights” is an order or request to switch off the lights

REGULAR Expressions

Regular Expression is a language that has been developed to help the user specify a search string to be found in the data. The pattern needs to be specified to get the desired results.

E.g. If a person wishes to find 10 numbers from the data but does not know the exact numbers, he/she could simply specify the pattern that those numbers are following to find them. We can divide the following regular expression into 2 parts which is the corpus and the pattern

Corpus: A Corpus is essentially a paragraph that is being specified under which an entire search will be executed. This has the entire bulk of data that can be used.

Pattern: This is the required pattern that can be entered in the search to obtain the required search results. Basic expressions that are used for the following are given below.

There are a few basic properties of a regular expression that need to be understood to use ‘RegEx’ (Regular Expression) efficiently.

1. Brackets (“[ ]”)

Brackets are used to specify a certain disjunction of characters. For instance, the bracket being used on a regular ‘s’ or ‘S’ can be used to return a capital or lower case ‘S’.
E.g. /[sS]mall = Small or small (any of the above combinations)
/[a b c] = ‘a’, ‘b’ or ‘c’ (any of the letters in the bracket)
/[0,1,2,3,4,5] = any digit from the bracket

2. Dash (“-”)
Dash is a regular expression that can specify the range for a particular search. If we put a dash between two letters it would match all the letters between them.
E.g. /[A-Z] = All upper-case letters between A and Z would match the search.
/[a-z] = All lower-case letters between a and z would match the search

3. Caret (“^”)
As normal syntax, this sign is also used to indicate a negation in the search that is conducted using this regular expression. It could also simply be used as a ‘caret’ symbol.
E.g. /[^A-Z]mall = Not any upper-case would be selected
/[^Dd] = This would indicate the search is for neither ‘d’ nor ‘D’
[e^] = Used as a simple caret simple as there is nothing to negate after ‘e’

4. Question Mark (“?”)
The question mark symbol is essentially used to display the optionality for the search that has been conducted in the previous search. Putting a “?” before a certain letter would indicate that the user would like to return the search with an option consisting of that letter and one without that letter.
E.g. /[malls?] = This would return either “mall” or “malls”
/[colou?r] = This would return either “color” or “colour”

5. Period(“*”)
The asterisk symbol “*” is used to match a certain character that is between an expression in the search. If the user has an expression that would make sense after entering another letter this period mark is very useful.
E.g. /[beg*n] = Will give out any word that makes sense like ‘begin’

6. Anchors
Anchors can be used to match a certain assertion in the text that has been searched.
E.g. Consider “^” and “$” as specified anchors. Then
^The = Matches a string that starts with ‘The’
bye$ = Matches a string that ends with ‘bye’

7. Character Classes
“\d”: Matches a single character which is a digit.

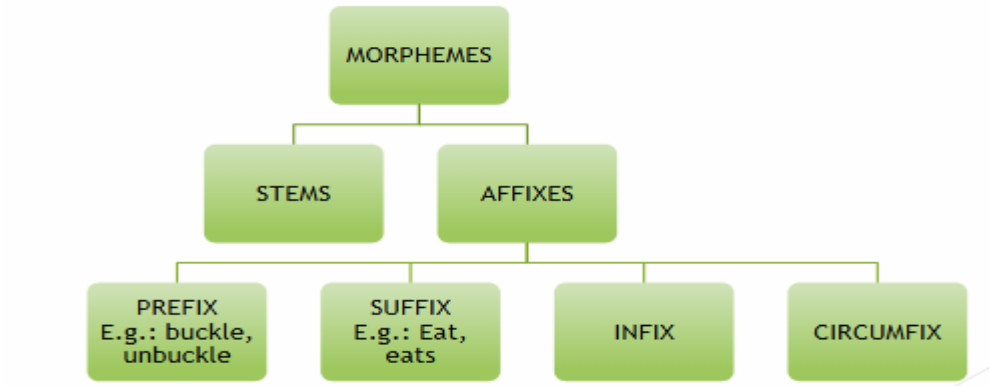
"\w": Matches a word character. This could be alphanumeric as well as with an underscore. "\s": Matches a whitespace character. This includes line breaks as well as tabs.
".": Matches any character.

MORPHOLOGY

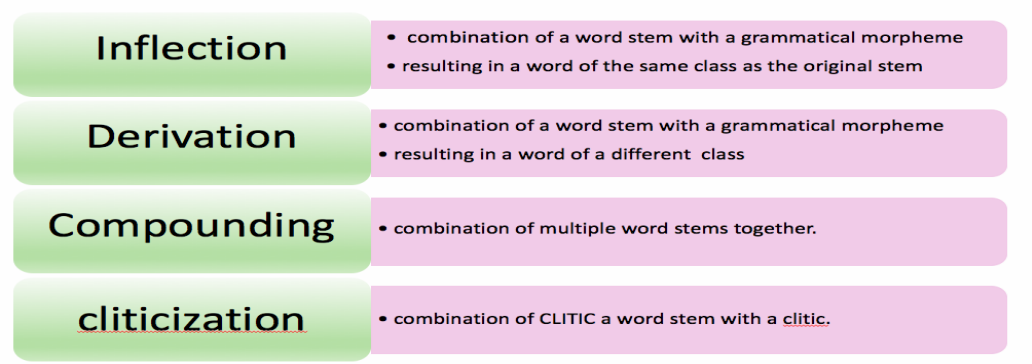
- Morphology is the domain of linguistics that analyzes the internal structure of words
- According to the classical approach in linguistics, words are formed of morphemes, which are the minimal (that is, non-decomposable) linguistic units that carry meaning

Why?

- Many language processing applications need to extract the information encoded in the words
- Parsers which analyze sentence structure need to know/check agreement between
  - Subjects and verbs
  - Adjectives and nouns
- Information retrieval systems benefit from know what the stem of a word is
- Machine translation systems need to analyze words to their components and generate words with specific features in the target language
- The word CATS contain two morphmenes: the morpheme cat and the morpheme -s. As this example suggests, it is often useful to distinguish two broad classes of morphemes: stems and affixes.
- The stem is the "main" morpheme of the word, supplying the main meaning, while the affixes add "additional" meanings of various kinds. 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, the word eats is composed of a stem eat and the suffix -s . The word unbuckle is composed.
- A word can have more than one affix. For example, the word rewrites has the prefix re-, the stem write, and the suffix -s. There are many ways to combine morphemes to create words.
- Four of these methods are common and play important roles in speech and language processing: inflection, derivation, compounding, and cliticization.



i) Inflectional Morphology

- **Phenomena of declination and conjugation (change of number, gender, time, person, mode and case).**
- It does not change the word class
  - Horse Horse**s**
  - Eat Eat**ing**
  - Like Like**s**
- English has a relatively simple inflectional system; only nouns, verbs, and sometimes adjectives can be inflected, and the number of possible inflectional affixes is quite small.**English nouns have only two kinds of inflection: an affix that marks plural and an affix that marks possessive.**

	Regular Noun		Irregular Noun	
Singular	**Cat**	**Fox**	**Mouse**	**OX**
Plural	cats	Foxes	Mice	Oxen

- The regular plural is spelled -s after most nouns. The possessive suffix is realized by apostrophe + -s for regular singular nouns ((children's).
- First, English has three kinds of verbs; main verbs , (eat, sleep, impeach), modal verbs (can, will, should), and primary verbs (be, have, do) . These verbs are called regular because just by knowing the stem we can predict the other forms by adding one of three predictable endings and making some regular spelling changes. The irregular verbs are those that have some more or less idiosyncratic forms of inflection.

Morphological class	Regular verb		Irregular verb	
Stem	**walk**	**try**	**eat**	**catch**
-s form	walks	tries	eats	catches
-ing form	walking	trying	eating	catching
Past form –ed form	walked	tried	ate	caught

ii)Derivational Morphology

derivation is the combination of a word stem with a grammatical morpheme, usually resulting in a word of a different class, often with a meaning hard to predict exactly. A very common kind of derivation in English is the formation of new nouns, often from verbs or adjectives. This process is called nominalization.

Suffix	Base verb/Adj	Noun
-tion	Computerize	Computerization
-ee	employ	employee
-er	kill	Killer
-ness	Lazy(A)	laziness

Adjectives can also be derived from nouns and verbs.

Suffix	Base word	Adjective
-al	Computation	Computational
-able	trace	traceable
-less	clue	clueless
-ly	man	manly

Compuding eg : eggplant , lemongrass , snowman

Cliticization

its status lies between a word and an affix.

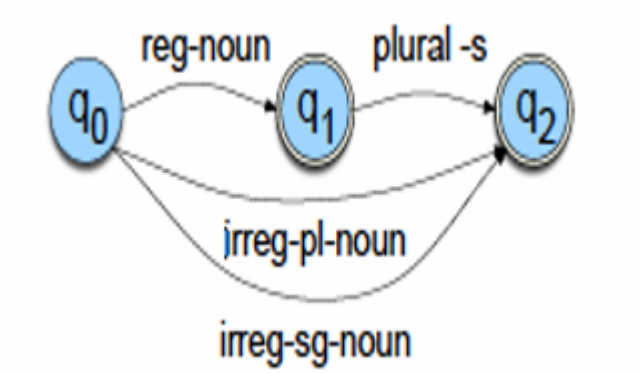
Full form	clitic
am	‘m
have	‘ve
would	‘d
will	‘ll

MORPHOLOGICAL ANALYSIS (Morphological Parsing)

In order to build a morphological parser, we’ll need at least the following:

1. **lexicon**: the list of stems and affixes, together with basic information about them (whether a stem is a Noun stem or a Verb stem, etc.).
2. **morphotactics**: the model of morpheme ordering that explains which classes of morphemes can follow other classes of morphemes inside a word. For example, the fact that the English plural morpheme follows the noun rather than preceding it is a morphotactic fact.
3. **orthographic rules**: these spelling rules are used to model the changes that occur in a word, usually when two morphemes combine (e.g., the y→ie spelling rule discussed above that changes city + -s

English	
Input	Morphologically Parsed Output
cats	cat +N +PL
cat	cat +N +SG
cities	city +N +Pl
geese	goose +N +Pl
goose	goose +N +Sg
goose	goose +V
gooses	goose +V +1P +Sg
merging	merge +V +PresPart
caught	catch +V +PastPart
caught	catch +V +Past



The FSA in the figure assumes that the lexicon includes regular nouns (reg-noun) that take the regular -s plural. The lexicon also includes irregular noun forms that don't take -s, both singular irreg-sg-noun (goose, mouse) and plural irreg-pl-noun (geese, mice).

FINITE-STATE TRANSDUCERS

A transducer maps between one representation and another; a finite-state transducer or FST is a type of finite automaton which maps between two sets of symbols. An FST defines a relation between sets of strings.

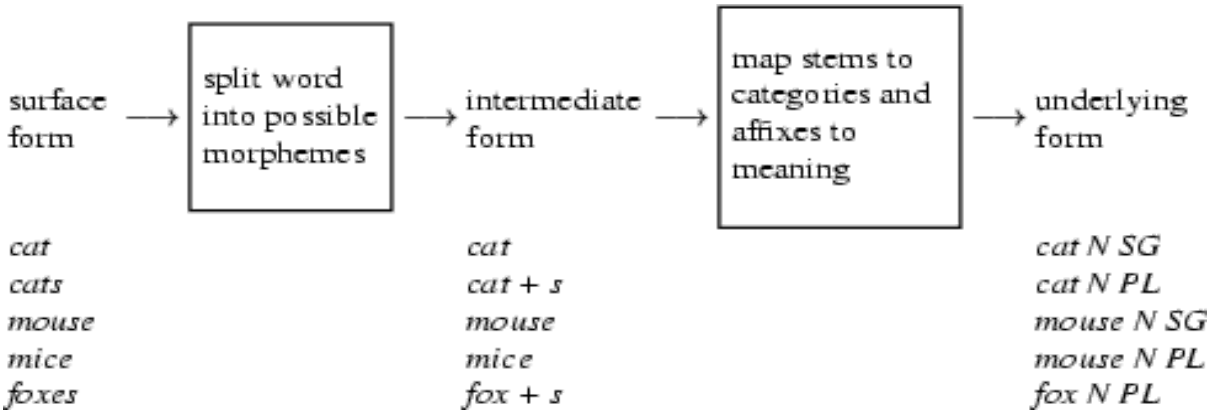
Let's begin with a formal definition. An FST can be formally defined with 7 parameters:

Q a finite set of N states q0,q1, . . . ,qN−1
S a finite set corresponding to the input alphabet
D a finite set corresponding to the output alphabet
q0 ∈ Q the start state
F ⊆ Q the set of final states
d(q,w) the transition function or transition matrix between states; Given a state q ∈ Q and a string w ∈ S∗ , d(q,w) returns a set of new states Q′ ∈ Q . d is thus a function from QÅ~S

s(q,w) the output function giving the set of possible output strings for each state and input. Given a state q ∈ Q and a string w ∈ S∗ , s(q,w) gives a set of output strings, each a string o ∈ D∗ . s is thus a function from QÅ~S∗ to 2D∗

FSTs have two additional closure properties.

- inversion: The inversion of a transducer T (T−1) simply switches the input and output labels. Thus if T maps from the input alphabet I to the output alphabet O, T^{-1} maps from O to I.
- composition: If T1 is a transducer from I1 to O1 and T2 a transducer from O1 to O2, then T1 ◦ T2 maps from I1 to O2.

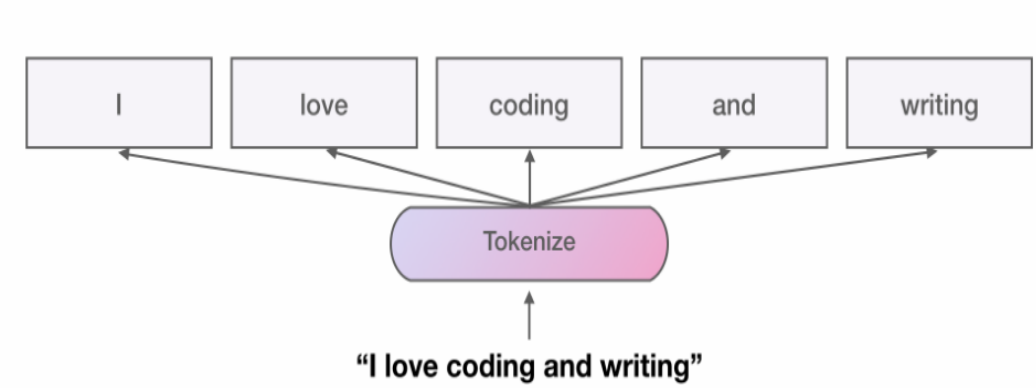


Morphological analyzer aids in tasks like tokenization, stemming, and lemmatization, which are crucial for text processing and information retrieval

TOKENIZATION :

When we collect the data, it is almost always in the form of long, meaningful sentences containing words, special characters, and whitespaces. For a human being, it is easier to read the sentences, but for an algorithm, the sentence is often of no use, as most of the NLP algorithms work on the words rather than sentences. Therefore, it is necessary to convert the sentences or documents to words before feeding them to the algorithm.

Tokenization, as the name suggests, is the process of converting sentences or documents into a set of tokens or words. In a sentence, the words are often separated by whitespaces but can also be separated by special characters such as a comma, a hyphen, etc. In the tokenization process, we would break the sentences into words and store them as a list of words rather than a continuous sentence.



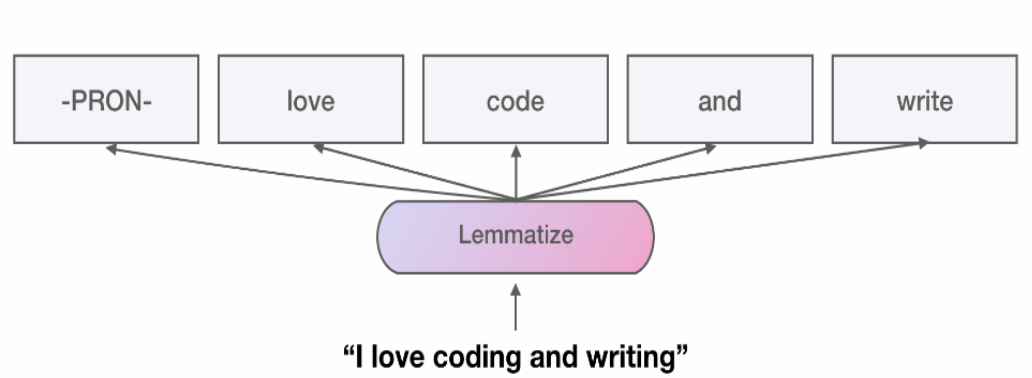
STEMMING

•Stemming is a technique used to extract the base form of the words by removing affixes from them.
•Stemming is a rule-based approach because it slices the inflected words from prefix or suffix as per the need.
•There are mainly two errors that *occur while performing Stemming, Over-stemming, and Under-stemming*.

A computer program or subroutine that stems word may be called a stemming program, stemming algorithm, or stemmer.
•**Porter Stemmer** uses suffix striping to produce stems. It does not follow the linguistic set of rules to produce stem for phases in different cases, due to this reason porter stemmer does not generate stems, i.e. actual English words.

LEMMATIZATION

It is the process where we take individual tokens from a sentence and we try to reduce them to their base form. The process that makes this possible is having a vocabulary and performing morphological analysis to remove inflectional endings. The output of the lemmatization process (as shown in the figure above) is the lemma or the base form of the word. For instance, a lemmatization process reduces the inflections, "am", "are", and "is", to the base form, "be". Take a look at the figure above for a full example and try to understand what it's doing.



	Stemming	Lemmatization
	Stemming is faster because it chops words without knowing the context of the word in given sentences.	Lemmatization is slower as compared to stemming but it knows the context of the word before proceeding.
	It is a rule-based approach.	It is a dictionary-based approach.
	Accuracy is less.	Accuracy is more as compared to Stemming.
	When we convert any word into root-form then stemming may create the non-existence meaning of a word.	Lemmatization always gives the dictionary meaning word while converting into root-form.
	Stemming is preferred when the meaning of the word is not important for analysis. Example: Spam Detection	Lemmatization would be recommended when the meaning of the word is important for analysis. Example: Question Answer
	For Example: "Studies" => "Studi"	For Example: "Studies" => "Study"

TOKENIZATION

Example

nltk.download('all')
import nltk

SENTENCE TOKENIZATION

```
from nltk.tokenize import sent_tokenize
text = "Good morning! Welcome to NLP practice session. It will be of great fun!"
sent_tok=sent_tokenize(text)
print(sent_tok)
```

['Good morning!', 'Welcome to NLP practice session.', 'It will be of great fun!']

CWORDTOKENIZATION

```
from nltk.tokenize import stopwords
text = "Good morning! Welcome to NLP practice session. It will be of great fun!"
word_tok = word_tokenize(text)
print(word_tok)
text_no_sw = [word for word in word_tok if not word in stopwords.words()]
print(text_no_sw)
```

['Good', 'morning', '!', 'Welcome', 'to', 'NLP', 'practice', 'session', '.', 'It', 'will', 'be', 'of', 'great', 'fun', '!']
['Good', 'morning', '!', 'Welcome', 'NLP', 'practice', 'session', '.', 'It', 'great', 'fun', '!']

STEMMING

```
from nltk.tokenize import word_tokenize
from nltk.stem.porter import PorterStemmer
porter = PorterStemmer()
text = "study studies studied cry cries crying cried"
tokenize = word_tokenize(text)
for w in tokenize:
  print("Stemming for {} is {}".format(w, porter.stem(w)))
```

Stemming for study is studi
Stemming for studies is studi
Stemming for studied is studi
Stemming for cry is cri
Stemming for cries is cri
Stemming for crying is cri
Stemming for cried is cri

LEMMATIZATION

```
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
lem = WordNetLemmatizer()
text = "study studies studied cry cries crying cried"
tokenize = word_tokenize(text)
```

```
for w in tokenize:
  print("Lemma for {} is {}".format(w, lem.lemmatize(w)))


Lemma for study is study
Lemma for studies is study
Lemma for studied is studied
Lemma for cry is cry
Lemma for cries is cry
Lemma for crying is cry
Lemma for cried is cried
```

FEATURE EXTRACTION :
TERM FREQUENCY AND INVERSE TERM FREQUENCY

Feature Extraction

Feature extraction entails mapping the textual data to real-valued vectors. It is a representation of analyzing text. It does not, however, represent the word sequences or positions.

Term Frequency - Inverse Document Frequency (TF-IDF) is a widely used statistical method in natural language processing and information retrieval. It measures how important a term is within a document relative to a collection of documents (i.e., relative to a corpus).

Words within a text document are transformed into importance numbers by a text vectorization process. There are many different text vectorization scoring schemes, with TF-IDF being one of the most common.

As its name implies, TF-IDF vectorizes/scores a word by multiplying the word's Term Frequency (TF) with the Inverse Document Frequency (IDF).

Term Frequency: TF of a term or word is the number of times the term appears in a document compared to the total number of words in the document.

$$TF = \frac{\text{number of times the term appears in the document}}{\text{total number of terms in the document}}$$

Inverse Document Frequency: IDF of a term reflects the proportion of documents in the corpus that contain the term. Words unique to a small percentage of documents (e.g., technical jargon terms) receive higher importance values than words common across all documents (e.g., a, the, and).

$$IDF = log(\frac{\text{number of the documents in the corpus}}{\text{number of documents in the corpus contain the term}})$$

The TF-IDF of a term is calculated by multiplying TF and IDF scores.

TF-IDF=TF*IDF

Translated into plain English, importance of a term is high when it occurs a lot in a given document and rarely in others. In short, commonality within a document measured by TF is balanced by rarity between documents measured by IDF. The resulting TF-IDF score reflects the importance of a term for a document in the corpus.

TF-IDF is useful in many natural language processing applications. For example, Search Engines use TF-IDF to rank the relevance of a document for a query. TF-IDF is also employed in text classification, text summarization, and topic modeling.

Note that there are some different approaches to calculating the IDF score. The base 10 logarithm is often used in the calculation. However, some libraries use a natural logarithm. In addition, one can be added to the denominator as follows in order to avoid division by zero.

$$IDF = log(\frac{\text{number of the documents in the corpus}}{\text{number of documents in the corpus contain the term} + 1})$$

Numerical Example

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:

$$TF = \frac{20}{100} = 0.2$$

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

$$IDF = log\frac{10000}{100} = 2$$

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

$$TF\text{-}IDF = 0.2 * 2 = 0.4$$

Python Implementation

Some popular python libraries have a function to calculate TF-IDF. The popular machine learning library `Sklearn` has `TfidfVectorizer()` function (docs).

We will write a TF-IDF function from scratch using the standard formula given above, but we will not apply any preprocessing operations such as stop words removal, stemming, punctuation removal, or lowercasing. It should be noted that the result may be different when using a native function built into a library.

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

First, let's construct a small corpus.

```
corpus = ['data science is one of the most important fields of science',
          'this is one of the best data science courses',
          'data scientists analyze data' ]
```

Next, we'll create a word set for the corpus:

```
words_set = set()

for doc in  corpus:
    words = doc.split(' ')
    words_set = words_set.union(set(words))

print('Number of words in the corpus:',len(words_set))
print('The words in the corpus: \n', words_set)
```

```
Number of words in the corpus: 14
The words in the corpus:
 {'important', 'scientists', 'best', 'courses', 'this', 'analyze', 'of', 'most', 'the',
```

```
ourses', 'this', 'analyze', 'of', 'most', 'the', 'is', 'science', 'fields', 'one', 'data'}
```

Computing Term Frequency

Now we can create a dataframe by the number of documents in the corpus and the word set, and use that information to compute the **term frequency (TF)**:

```
n_docs = len(corpus)          # Number of documents in the corpus
n_words_set = len(words_set) # Number of unique words in the

df_tf = pd.DataFrame(np.zeros((n_docs, n_words_set)), columns=words_set)

# Compute Term Frequency (TF)
for i in range(n_docs):
    words = corpus[i].split(' ') # Words in the document
    for w in words:
        df_tf[w][i] = df_tf[w][i] + (1 / len(words))

df_tf
```

	important	scientists	best	courses	this	analyze	of
0	0.090909	0.00	0.000000	0.000000	0.000000	0.00	0.181818
1	0.000000	0.00	0.111111	0.111111	0.111111	0.00	0.111111
2	0.000000	0.25	0.000000	0.000000	0.000000	0.25	0.00000

The dataframe above shows we have a column for each word and a row for each document. This shows the frequency of each word in each document.

most	the	is	science	fields	one	data
0.090909	0.090909	0.090909	0.181818	0.090909	0.090909	0.090
0.000000	0.111111	0.111111	0.111111	0.000000	0.111111	0.11111
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.500

Computing Inverse Document Frequency

Now, we'll compute the **inverse document frequency (IDF)**:

```
print("IDF of: ")

idf = {}

for w in words_set:
    k = 0    # number of documents in the corpus that contain this word

    for i in range(n_docs):
        if w in corpus[i].split():
            k += 1

    idf[w] =  np.log10(n_docs / k)

    print(f'{w:>15}: {idf[w]:>10}' )
```

```
IDF of:
    important: 0.47712125471966244
   scientists: 0.47712125471966244
         best: 0.47712125471966244
      courses: 0.47712125471966244
         this: 0.47712125471966244
      analyze: 0.47712125471966244
           of: 0.17609125905568124
         most: 0.47712125471966244
          the: 0.17609125905568124
           is: 0.17609125905568124
      science: 0.17609125905568124
       fields: 0.47712125471966244
          one: 0.17609125905568124
         data:          0.0
```

Putting it Together: Computing TF-IDF

Since we have TF and IDF now, we can compute **TF-IDF**:

```
df_tf_idf = df_tf.copy()

for w in words_set:
    for i in range(n_docs):
        df_tf_idf[w][i] = df_tf[w][i] * idf[w]

df_tf_idf
```

	important	scientists	best	courses	this	analyze	of
0	0.043375	0.00000	0.000000	0.000000	0.000000	0.00000	0.0
1	0.000000	0.00000	0.053013	0.053013	0.053013	0.00000	0.0
2	0.000000	0.11928	0.000000	0.000000	0.000000	0.11928	0.0

Notice that "data" has an IDF of 0 because it appears in every document. As a result, is not considered to be an important term in this corpus. This will change slightly in the following sklearn implementation, where "data" will be non-zero.

Parts of Speech Tagging

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

Part of speech	Role	Examples
Noun	Is a person, place, or thing	mountain, bagel, Poland
Pronoun	Replaces a noun	you, she, we
Adjective	Gives information about what a noun is like	efficient, windy, colorful
Verb	Is an action or a state of being	learn, is, go
Adverb	Gives information about a verb, an adjective, or another adverb	efficiently, always, very
Preposition	Gives information about how a noun or pronoun is connected to another word	from, about, at
Conjunction	Connects two other words or phrases	so, because, and
Interjection	Is an exclamation	yay, ow, wow

- Some sources also include the category **articles** (like "a" or "the") in the list of parts of speech, but other sources consider them to be adjectives. NLTK uses the word **determiner** to refer to articles.

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	+,%, &
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	*$*
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	#
PDT	Predeterminer	*all, both*	"	Left quote	' or "
POS	Possessive ending	*'s*	"	Right quote	' or "
PRP	Personal pronoun	*I, you, he*	(	Left parenthesis	[, (, {, <
PRP$	Possessive pronoun	*your, one's*	)	Right parenthesis	],), }, >
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	. ! ?
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	: ; ... – -
RP	Particle	*up, off*			

ENGLISH WORD CLASSES

Parts-of-speech can be divided into two broad supercategories: **closed class typesand open class types**.

Closed classes are those that **have relatively fixed membership**. For example, prepositions are a closed class because there is a fixed set of them in English; new prepositions are rarely coined.

By contrast nouns and verbs are **open classes because new nouns and verbs are continually coined or borrowed from other languages**.

Closed class words are also generally function words like of, it, and, or you, which tend to be very short, occur frequently, and often have structuring uses in grammar.

There are four major open classes that occur in the languages of the world; nouns,
verbs, adjectives, and adverbs. The closed class words includes the following

- **prepositions:** on, under, over, near, by, at, from, to, with
- **determiners:** a, an, the
- **pronouns:** she, who, I, others
- **conjunctions:** and, but, or, as, if, when
- **auxiliary verbs:** can, may, should, are
- **particles:** up, down, on, off, in, out, at, by,
- **numerals:** one, two, three, first, second, third

Most tagging algorithms fall into one of two classes: **rule-based taggers and stochastic taggers taggers.**
Rule-based taggers generally involve a large database of hand-written disambiguation rules which specify, for example, that an ambiguous word is a noun rather than a verb if it follows a determiner.
Stochastic taggers generally resolve tagging ambiguities by using a training corpus to compute the probability of a given word having a given tag in a given context.

RULE-BASED PART-OF-SPEECH TAGGING

The earliest algorithms for automatically assigning part-of-speech were based on a two stage
architecture.

- The first stage used a dictionary to assign each word a list of potential parts-of-speech.
- The second stage used large lists of hand-written disambiguation rules to win now down this list to a single part-of-speech for each word.

In the first stage of the tagger, each word is run through the two-level lexicon transducer and the entries for all possible parts-of-speech are returned. Then it applies a large set of constraints (as many as 3,744 constraints in
the EngCG-2 system) to the input sentence to rule out incorrect parts-of-speech.

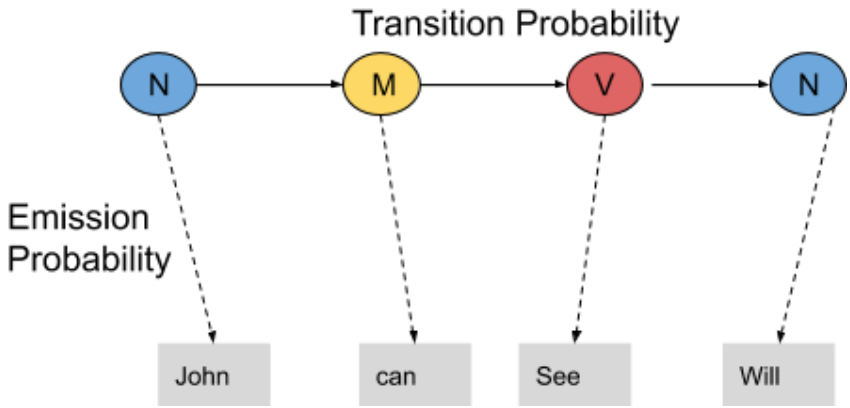
Example: •If an ambiguous/unknown word X is preceded by a determiner and followed by a noun, tag it as an adjective.

HMM PART-OF-SPEECH TAGGING

Bayesian inference or Bayesian classification was applied successfully to language problems as early as the late 1950s, Bayes' rule gives us a way to break down any conditional probability P(x|y) into three other probabilities:
$P(x|y) = P(y|x)/(P(x)P(y))$
HMM taggers therefore make two simplifying assumptions. The first assumption is that the probability of a word appearing is dependent only on its own part-of-speech tag; that it is independent of other words around it, and of the other tags around it:
$P(wn1|tn1) \approx \Pi^{n}_{k=1} P(wi|ti)$
The second assumption is that the probability of a tag appearing is dependent only
on the previous tag, the bigram assumption.
$P(tn1) \approx \Pi^{n}_{k=1=1} P(ti|ti-1)$

A Hidden Markov Model (HMM) allows us to talk about both observed events (like words that we see in the input) and hidden events
An HMM is specified by the following components:
Q = q1q2 . . .qN a set of N states
A = a11a12 . . .an1 . . .ann
a transition probability matrix A , each ai j representing the probability of moving from state i to state j ,
O = o1o2 . . .oT a sequence of T observations , each one drawn from a vocabulary V= v1,v2, ...,vV .
A sequence of observation likelihoods: , also called emission probabilities , each expressing the probability of an observation ot being generated from a state i .
q0,qF a special start state and end (final) state

An HMM thus has two kinds of probabilities; the A transition probabilities, and the B observation likelihoods, corresponding respectively to the prior and likelihood probabilities.



The Viterbi Algorithm for HMM Tagging

For any model, such as an HMM, that contains hidden variables, the task of determining
which sequence of variables is the underlying source of some sequence of observations is called the decoding task. The Viterbi algorithm is perhaps the most common decoding algorithm used for HMMs, whether for part-of-speech tagging or for speech recognition.

TRANSFORMATION-BASED TAGGING

Transformation-Based Tagging, sometimes called Brill tagging, is an instance of the Transformation-Based Learning (TBL) approach to machine learning and draws inspiration from both the rule-based and stochastic taggers.

Like the rulebased taggers, TBL is based on rules that specify what tags should be assigned to what words. But like the stochastic taggers, TBL is a machine learning technique, in which rules are automatically induced from the data. Like some but not all of the HMMtaggers, TBL is a supervised learning technique; it assumes a pre-tagged training corpus.

race is most likely to be a noun:

P(NN|race) = .98

P(VB|race) = .02

This means that the two examples of race will be coded as NN.

It first labels every word with its mostlikely tag. It then examines every possible transformation, and selects the one that results in the most improved tagging. Finally, it then re-tags the data according to this rule. The last two stages are repeated until some stopping criterion is reached, such as insufficient improvement over the previous pass. Note that stage two requires that TBL knows the correct tag of each word; that is, TBL is a supervised learning algorithm.

```
import nltk
import string
text = "This is an example text for stopword
```

```
removal and filtering. This is done using
NLTK's stopwords."
words = nltk.word_tokenize(text)
stopwords =
nltk.corpus.stopwords.words("english")

# Extending the stopwords list
stopwords.extend(string.punctuation)

# Remove stop words and tokens with length <
2
cleaned = [word.lower() for word in words if
(word not in stopwords) and len(word) > 2]
""" End of stolen code """

# Assign POS Tags to the words
tagged = nltk.pos_tag(cleaned)
print(tagged)

[("this", "DT"), ("example", "NN"), ("text",
"NN"), ("stopword", "NN"), ("removal",
"NN"), ("filtering", "VBG"), ("this", "DT"),
("done", "VBN"), ("using", "VBG"), ("nltk",
"JJ"), ("stopwords", "NNS")]

from nltk.corpus import stopwords
text = "They refuse to permit us to obtain the
refuse permit!"
word_tok = word_tokenize(text)
#print(word_tok)
text_no_sw = [word for word in word_tok if
not word in stopwords.words()]
#print(text_no_sw)
print(nltk.pos_tag(text_no_sw))

[('They', 'PRP'), ('refuse', 'VBP'), ('permit',
'VBP'), ('us', 'PRP'), ('obtain', 'VB'), ('refuse',
'JJ'), ('permit', 'NN'), ('!', '.')]
```

What is Named Entity Recognition?

Named Entity Recognition (NER) is a fundamental task in Natural Language Processing (NLP) that involves locating and classifying named entities mentioned in unstructured text into predefined categories such as names, organizations, locations, dates, quantities, percentages, and monetary values. NER serves as a foundational component in various NLP applications, including

information extraction, question answering, machine translation, and sentiment analysis.

At its core, NER processes textual data to identify and categorize key information. For example, in the sentence

"Apple is looking at buying U.K. startup for $1 billion."

An NER system should recognize "Apple" as an Organization (ORG), "U.K." as a Geopolitical entity (GPE), and "$1 billion" as a Monetary value (MONEY).

Labels in NER

In NER, labels are the categories assigned to words or phrases identified as named entities within a piece of text. These labels indicate the type of entity detected, such as a person, organization, location, or date. The labeling process allows unstructured text to be converted into structured data, which can be used for various applications like information retrieval, question answering, and data analysis.

The set of labels used in NER can vary depending on the specific application, domain, or dataset. However, some standard labels are widely used across different NER systems:

Labels	**Description**	**Example**
Person (PER)	Names of people or fictional characters.	Albert Einstein," "Marie Curie," "Sherlock Holmes."
Organization (ORG)	Names of companies, institutions, agencies, or other groups of people.	"Google," "United Nations," "Harvard University."
Location (LOC)	Names of geographical places such as cities, countries, mountains, rivers.	"Mount Everest," "Nile River," "Paris."
Geo-Political Entity (GPE)	Geographical regions that are also political entities.	"United States," "Germany," "Tokyo."
Date	Expressions of calendar dates or periods.	"January 1, 2022," "the 19th century," "2010-2015."
Time	Specific times within a day or durations.	"5 PM," "midnight," "two hours."
Money	Monetary values, often accompanied by currency symbols.	"$100," "€50 million," "1,000 yen."
Percent	Percentage expressions.	"50%," "3.14%," "half."

Facility (FAC)	Buildings or infrastructure.	"Eiffel Tower," "JFK Airport," "Golden Gate Bridge."
Product	Objects, vehicles, software, or any tangible items.	"iPhone," "Boeing 747," "Windows 10."
Event	Named occurrences such as wars, sports events, disasters.	"World War II," "Olympics," "Hurricane Katrina."
Work of Art	Titles of books, songs, paintings, movies.	"Mona Lisa," "To Kill a Mockingbird," "Star Wars."
Language	Names of languages.	"English," "Mandarin," "Spanish."
Law	Legal documents, treaties, acts.	"The Affordable Care Act," "Treaty of Versailles."
NORP (Nationality, Religious, or Political Group)	Nationalities, religious groups, or political affiliations.	"American," "Christians," "Democrat."

Methods of NER

1. Dictionary-based

This is the simplest NER method. In this approach, a dictionary containing vocabulary is used. Basic string-matching algorithms check whether the entity is present in the given text against the items in the vocabulary. This method is generally not employed because the dictionary that is used is required to be updated and maintained consistently.

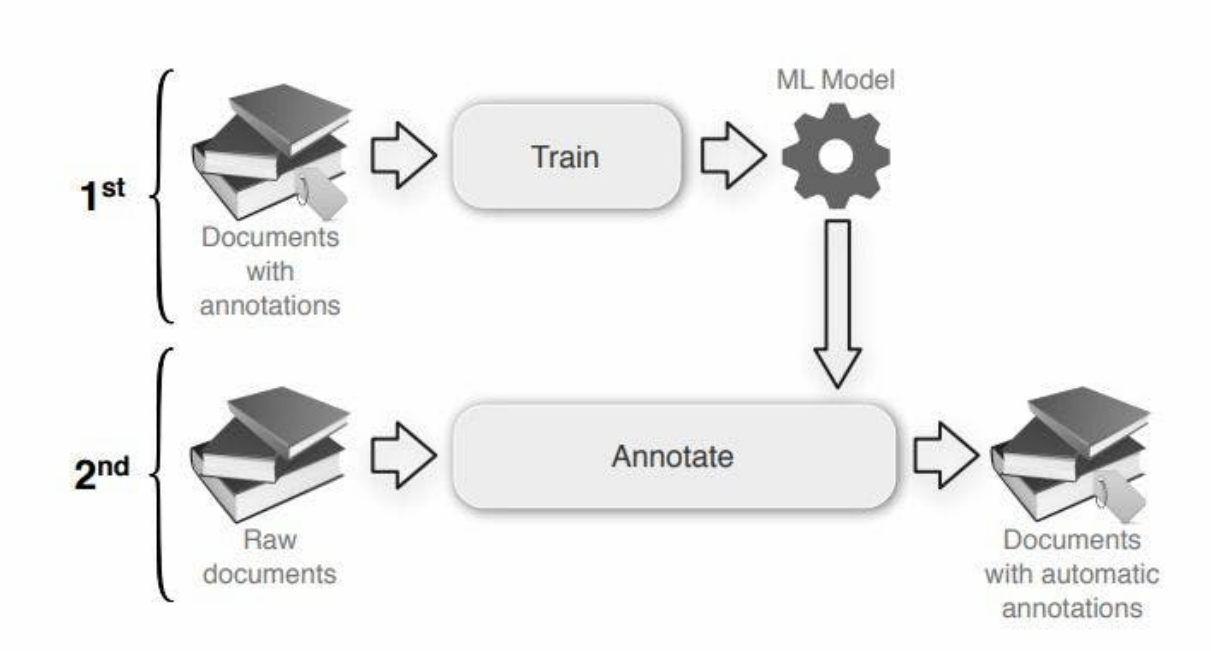
2. Rule-based

In this method, a predefined set of rules for information extraction is used which are pattern-based and context-based. Pattern-based rule uses the morphological pattern of the words while context-based uses the context of the word given in the text document.

3. Machine learning-based

This method solves a lot of limitations of the above two methods. It is a statistical-based model that tries to make a feature-based representation of the observed data. It can recognize an existing entity name even with small spelling variations.

The machine learning-based approach involves two phases for doing NER. The first phase trains the ML model on annotated documents. In the next phase, the trained model is used to annotate the raw documents. The process is similar to a normal ML model pipeline.



4. Deep learning-based

Deep learning NER is more accurate than the ML-based method because it is capable of assembling words, enabling it to understand the semantic and syntactic relationship between various words better. It is also capable of analyzing topic-specific and high-level words automatically.

How named entity recognition works

Humans can easily detect entities belonging to various categories like people, location, money, etc. For computers to do the same, they first need to recognize and then categorize them. NLP and machine learning are used to achieve this.

- **NLP:** Helps machines understand the rules of language and helps make intelligent systems that can easily derive meaning from text and speech.
- **Machine learning:** Helps machines learn and improve over time by using various algorithms and training data.

Any NER model has a two-step process: i) detect a named entity and ii) categorize the entity.

i) Detect a named entity

The first step for named entity recognition is detecting an entity or keyword from the given input text. The entity can be a word or a group of words.

ii) Categorize the entity

This step requires the creation of entity categories. Some common categories are:

Person - Cristiano, Sachin, Dhoni

Organization - Google, Microsoft, Visa

Time - 2006, 13:32

Location - India, Lucknow, USA

Percent - 99%

Money - 2 billion dollars

You can also create your own entity categories.

Good training data is essential for a model to categorize detected entities into a relevant predetermined entity. Therefore, it's vital to train the model correctly on suitable data.

N-Grams

- Models that assign probabilities to sequences of words are called language models (LMs).
- The simplest language model that assigns probabilities to sentences and sequences of words is n-gram language model.
- An n-gram is a sequence of N words:
- N-gram: An n-gram is a sequence of n words. They are of various types-

1. Unigrams (1-grams): These are single words, like individual building blocks of a sentence. -
   Example: In the sentence "I have a cat," the unigrams are "I," "have," "a," and "cat."
2. Bigrams (2-grams): These are pairs of words that appear next to each other in a sentence. -
   Example: In the sentence "I love ice cream," the bigrams are "I love," "love ice," and "ice cream."
3. Trigrams (3-grams): These are groups of three words in a row.
   Example: In the sentence "I want to learn," the trigrams are "I want to" and "want to learn."
4. 4-grams (Quadgrams or 4-tuples): These consist of four consecutive words. - Example: In the sentence "The quick brown fox jumps," a 4-gram could be "The quick brown fox" or "quick brown fox jumps."
5. 5-grams (5-tuples): These are sequences of five consecutive words.

We can use n-gram models to estimate the probability of the last word of an n-gram given the previous words, and also to assign probabilities to entire word sequences

Probabilistic Language Models

Probabilistic language models can be used to assign a probability to a sentence in many NLP tasks. – In many NLP applications, we can use the probability as a way to choose a better sentence or word over a less-appropriate one.

• Machine Translation:

– P(high winds tonight) > P(large winds tonight)

Spell Correction:

– Thek office is about ten minutes from here

– P(The office is) > P(Then office is)

• Speech Recognition:

– P(I saw a van) >> P(eyes awe of an)

• Summarization, question-answering, …

Our goal is to compute the probability of a sentence or sequence of words

W (=w_1 ,w_2 ,…,w_n): P(W) = P(w_1 ,w_2 ,…,w_n)

• What is the probability of an upcoming word?

: – P(w_5 | w_1 ,w_2 ,w_3 ,w_4)

• A model that computes either of these:

– P(W) or P(w_n | w_1 ,w_2 ,…w_{n-1}) is called a language model.

Chain Rule of Probability

How can we compute probabilities of entire word sequences like w_1 ,w_2 ,…,w_n?

– The probability of the word sequence w_1 ,w_2 ,…,w_n is P(w_1 ,w_2 ,…,w_n).

We can use the chain rule of the probability to decompose this probability:

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

$$= \prod_{k=1}^{n} P(w_k \mid w_1^{k-1})$$

Example:

P(the man from jupiter) = P(the) P(man|the) P(from|the man) P(jupiter|the man from)

The chain rule shows the link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words.

• Definition of Conditional Probabilities:

P(B|A) = P(A,B) / P(A) ➔ P(A,B) = P(A) P(B|A)

• Conditional Probabilities with More Variables: P(A,B,C,D) = P(A) P(B|A) P(C|A,B) P(D|A,B,C)

• **Chain Rule: P(w_1… w_n) = P(w_1) P(w_2 |w_1) P(w_3 |w_1w_2) … P(w_n |w_1…w_{n-1})**

Unigram(1-gram): **No history is used.**
Bi-gram(2-gram): **One word history**
Tri-gram(3-gram): **Two words history**
Four-gram(4-gram): **Three words history**
Five-gram(5-gram):**Four words history**

Estimating N-Gram Probabilities

Estimating n-gram probabilities is called maximum likelihood estimation (or MLE).
• We get the MLE estimate for the parameters of an n-gram model by getting counts from a corpus, and normalizing the counts so that they lie between 0 and 1.

Estimating bigram probabilities:

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

where C is the count of that pattern in the corpus

Bi-gram(2-gram): **One word history**

$$P(w_1, w_2) = \prod_{i=2} P(w_2 \mid w_1) \qquad P(w_i \mid w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

Tri-gram(2-gram): **Two words history**

$$P(w_1, w_2, w_3) = \prod_{i=3} P(w_3 \mid w_1, w_2) \qquad P(w_i \mid w_{i-1}, w_{i-2}) = \frac{\text{count}(w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-2}, w_{i-1})}$$

Four-gram(4-gram): **Three words history**

$$P(w_1, w_2, w_3, w_4) = \prod_{i=4} P(w_4 \mid w_1, w_2, w_3)$$

$$P(w_i \mid w_{i-1}, w_{i-2}, w_{i-3}) = \frac{\text{count}(w_{i-3}, w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-3}, w_{i-2}, w_{i-1})}$$

Estimating N-Gram probabilities

$$P(w_n \mid w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1} w_n)}{C(w_{n-N+1}^{n-1})}$$

Estimating N-Gram Probabilities Bigram Example :

A mini-corpus: We augment each sentence with a special symbol at the beginning of the sentence, to give us the bigram context of the first word, and special end-symbol .

<s>I am Sam </s>
<s>Sam I am </s>
<s>I fly </s>

• Unique words: I, am, Sam, fly

Bigrams: and are also tokens. There are 6(4+2) tokens and 6*6=36 bigrams

P(I|<s>)=2/3	**P(Sam|<s>)=1/3**	**P(am|<s>)=0**	**P(fly|<s>)=0**	P(<s>|<s>)=0	**P(</s>|<s>)=0**
P(I|I)=0	**P(Sam|I)=0**	**P(am|I)=2/3**	**P(fly|I)=1/3**	P(<s>|I)=0	**P(</s>|I)=0**
P(I|am)=0	**P(Sam|am)=1/2**	**P(am|am)=0**	**P(fly|am)=0**	P(<s>|am)=0	**P(</s>|am)=1/2**
P(I|Sam)=1/2	**P(Sam|Sam)=0**	**P(am|Sam)=0**	**P(fly|Sam)=0**	P(<s>|Sam)=0	**P(</s>|Sam)=1/2**
P(I|fly)=0	**P(Sam|fly)=0**	**P(am|fly)=0**	**P(fly|fly)=0**	P(<s>|fly)=0	**P(</s>|fly)=1**
P(I|</s>)=0	P(Sam|</s>)=0	P(am|</s>)=0	P(fly|</s>)=0	P(<s>|</s>)=1 or ?	P(</s>|</s>)=0

Exercise 1: Estimating Bi-gram probabilities

What is the most probable next word predicted by the model for the following word sequence?

Given Corpus

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

1) <S> Do ?

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Next word prediction probability W_{i-1}=do

Next word	Probability Next Word = $\frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$
P(</S> |do)	0/4
P(<I> | do)	**2/4**
P(<am>| do)	0/4
P(<Henry>| do)	1/4
P(<like | do)	1/4
P(<college | do)	0/4
P(do | do)	0/4

I is more probable

2) <S> I like Henry ?

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Next word prediction probability W_{i-1}=Henry

Next word	Probability Next Word= $\frac{N}{D} = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$
P(</S> | Henry)	**3/5**
P(<I> | Henry)	1/5
P(<am>| Henry)	0
P(<Henry>| Henry)	0
P(<like | Henry)	1/5
P(<college | Henry)	0
P(do | Henry)	0

</S> is more probable

3) <S> Do I like ?

Use Tri-gram

P<I like>=3

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Next word prediction probability

W_{i-2}=I and W_{i-1}=like

Next word	Probability Next Word= $\frac{\text{count}(w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-2}, w_{i-1})}$
P(</S> |I like)	0/3
P(<I> | I like)	0/3
P(<am>| I like)	0/3
P(<Henry>| I like)	1/3
P(<like | I like)	0/3
P(<college | I like)	2/3
P(do | I like)	0/3

College is probable

4) <S> Do I like college ?

Use Four-gram

Next word prediction probability

W_{i-3}=I, W_{i-2}=like W_{i-1}=college

Next word	Probability Next Word= $\frac{\text{count}(w_{i-3}, w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-3}, w_{i-2}, w_{i-1})}$
P(</S> |I like college)	2/2
P(<I> | I like college)	0/2
P(<am>| I like college)	0/2
P(<Henry>| I like college)	0/2
P(<like | I like college)	0/2
P(<college | I like college)	0/2
P(do | I like college)	0/2

</S> is more probable

Which of the following sentence is better. i.e. Gets a higher probability with this model. Use Bi-gram

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

1. <S> I like college </S>

<S> like college </S>=?

=P(I| <S>) × P(like | I) × P(college| like) × P(</S> | college)

=3/7 × 3/6 × 3/5 ×3/3 = 9/70=0.13

2. <S> Do I like Henry </S>

=P(do | <S>) × P(I | do) × P(like | I) × P(Henry | like) × P(</S> | Henry)

=3/7 × 2/4 × 3/6 ×2/5 ×3/5 = 9/350=0.0257

ANS: First statement is more probable

SMOOTHING

Smoothing What do we do with words that are in our vocabulary (they are not unknown words) but appear in a test set in an unseen context (for example they appear after a word they never appeared after in training)? To keep a language model from assigning zero probability to these unseen events, we'll have to shave off a bit of probability mass from some more frequent events and give it to the events we've never seen. This modification is called smoothing or discounting.
Smoothing: Taking a bit of the probability mass from more frequent events and giving it to unseen events,sometimes also called "discounting"

Many different smoothing techniques:
• Laplace (add-one) or additive
• Add-k
• Stupid backoff
• Kneser-Ney

Laplace Smoothing

- Add one to all n-gram counts before they are normalized into probabilities
- Not the highest-performing technique for language modeling, but a useful baseline
- Practical method for other text classification tasks

$$\bullet\ P(w_i) = \frac{c_i}{N} \rightarrow P_{\text{Laplace}}(w_i) = \frac{c_i+1}{N+V}$$

Example: Laplace Smoothing

Corpus Statistics:

Unigram	Frequency
Chicago	4
is	8
cold	6
hot	0

Bigram	Frequency
Chicago is	2
is cold	4
is hot	0
…	0

$P(w_i) = \frac{c_i}{N}$

Unigram	Probability
Chicago	$\frac{4}{18} = 0.22$
is	$\frac{8}{18} = 0.44$
cold	$\frac{6}{18} = 0.33$
hot	$\frac{0}{18} = 0.00$

Bigram	Probability
Chicago is	$\frac{2}{4} = 0.50$
is cold	$\frac{4}{8} = 0.50$
is hot	$\frac{0}{8} = 0.00$

After additing one and its probability

Example: Laplace Smoothing

Corpus Statistics:

Unigram	Frequency
Chicago	4+1
is	8+1
cold	6+1
hot	0+1

Bigram	Frequency
Chicago is	2+1
is cold	4+1
is hot	0+1
…	0+1

$P(w_i) = \frac{c_i}{N} \rightarrow P_{\text{Laplace}}(w_i) = \frac{c_i+1}{N+V}$

Unigram	Probability
Chicago	$\frac{5}{22} = 0.23$
is	$\frac{9}{22} = 0.41$
cold	$\frac{7}{22} = 0.32$
hot	$\frac{1}{22} = 0.05$

Bigram	Probability
Chicago is	
is cold	
is hot	

Add-K Smoothing

Moves a bit less of the probability mass from seen to unseen events • Rather than adding one to each count, add a fractional count

• 0.5

• 0.05

• 0.01

The value k can be optimized on a validation set

- $P(w_i) = \frac{c_i}{N} \rightarrow P_{\text{Add-K}}(w_i) = \frac{c_i+k}{N+kV}$

Kneser-Ney Smoothing

One of the most commonly used and best-performing n-gram smoothing methods

• Incorporates absolute discounting

- $P_{\text{AbsoluteDiscounting}}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)-d}{\sum_v C(w_{i-1}v)} + \lambda(w_{i-1})P(w_i)$

Discounted Bigram

Unigram with interpolation weight

Backoff

• If the n-gram we need has zero counts, approximate it by backing off to the (n-1)-gram • Continue backing off until we reach a size that has non-zero counts

• Just like with smoothing, some probability mass from higher order n-grams needs to be redistributed to lower-order ngrams