MANGALORE INSTITUTE OF TECHNOLOGY & ENGINEERING
(A Unit of Rajalaxmi Education Trust®, Mangalore)
Autonomous Institute affiliated to VTU, Belagavi, Approved by AICTE, New Delhi
Accredited by NAAC with A+ Grade & ISO 9001:2015 Certified Institution

Department of Artificial Intelligence & Machine Learning

Course Materials

Course Title : Natural Language Processing

Course Code: 23AIPC401

Scheme : 2023 (Autonomous)

By
Sunil Kumar S,
Dept of AI & ML,
MITE, Moodabidri

MANGALORE INSTITUTE OF TECHNOLOGY & ENGINEERING

(A Unit of Rajalaxmi Education Trust®, Mangalore)
Autonomous Institute affiliated to VTU, Belagavi, Approved by AICTE, New Delhi
Accredited by NAAC with A+ Grade & ISO 9001:2015 Certified Institution

Natural Language Processing			
Semester	**VII**	CIE Marks	**50**
Course Code	**23AIPC401**	SEE Marks	**50**
Teaching Hours/Week (L:T: P)	**3:0:2**	Exam Hrs	**3**
Total Hours	**40+24**	Credits	**4**

Course Learning Objectives: This course is designed to

1. **Impart** knowledge on linguistic structure, tokenization techniques, corpora, and text similarity measures.
2. **Provide** knowledge on statistical approaches to language modeling using N-gram models, smoothing, perplexity, and text generation.
3. **Familiarize** semantic representations such as embeddings and neural network architectures used for NLP tasks.
4. **Provide knowledge on** modern Large Language Models and Transformer architectures, including prompting, generation, training principles, and ethical considerations

Module 1: Introduction — **No. of Hrs: 8+6**

Introduction, Language and Knowledge, Challenges of NLP, Language and Grammar,NLP Applications, Words and Tokens: Words, Morphemes: Parts of Words, Unicode, Subword Tokenization: Byte-Pair Encoding, Rule-based tokenization, Regular Expressions, Minimum Edit Distance.

Laboratory Experiments :

1. Write a Python program to perform following tasks
   a. Read a text corpus.
   b. Perform word tokenization and sentence tokenization.
   c. Apply Byte Pair Encoding (BPE) for subword tokenization.
   d. Display the generated subword vocabulary.
2. Write a python program to perform following tasks
   a. Uses regular expressions to extract Email IDs, URLs, Dates from text.
   b. Implement Minimum Edit Distance algorithm to compute distance between two words.
   c. Display edit operations (insert, delete, substitute).

Text Book 1: Chapter 1.1,1.3 ,1.4,1.5,1.7
Text Book 2: Chapter 2:2.1-2.5,2.7,2.9

Module 2 N-gram Language Models — **No. of Hrs: 8+4**

N-Grams, Evaluating Language Models: Training and Test Sets, Evaluating Language Models: Perplexity, Sampling sentences from a language model, Smoothing Interpolation, and Backoff.

Laboratory Experiments :

1. Write a python program to perform following tasks
    a. Preprocess a text corpus.
    b. Generate unigram and bigram counts.
    c. Compute probability of next word using bigram model.
2. Write a python program to perform following tasks
    a. Train an N-gram language model.
    b. Split the dataset into training and test sets.
    c. Compute perplexity of the test dataset.

Text book 2: Chapter 3:3.1 to 3.4,3.6

Module 3 Embeddings &Classification	**No. of Hrs: 8+6**

Embedding: Lexical Semantics, Vector Semantics, Simple count-based embeddings, Cosine for measuring similarity, Word2vec, Evaluating Vector Models. Feedforward networks for NLP: Classification, Embeddings as the input to neural net classifiers, Training Neural Nets.

Laboratory Experiments :

1. Write a python program to perform following tasks:
    a. Load pretrained Word2Vec or GloVe embeddings.
    b. Accept two words from the user.
    c. Compute cosine similarity.
    d. Display top 5 similar words.
2. Write a python program to perform following tasks:
    a. Convert text dataset into embeddings.
    b. Train a feedforward neural network classifier.
    c. Classify documents into categories.
    d. Evaluate using accuracy and confusion matrix.

Text Book 2: Chapter 5:5.1 to 5.5,5.9 Chapter 6:6.3-6.6

Module 4 :Large Language Models	**No. of Hrs: 8+4**

Large Language Models, Three architectures for language models, Conditional Generation of

Text: The Intuition, Prompting, Generation and Sampling, Training Large Language Models, Evaluating Large Language Models.

Laboratory Experiments :

1. Develop a Python program using HuggingFace Transformers to:
    a. Load a pretrained GPT-based model.
    b. Accept a prompt from user.
    c. Generate text continuation.
    d. Compare outputs using different sampling methods.
2. Write a program to:
    a. Generate multiple responses using an LLM.
    b. Compute evaluation metrics such as:
        i. Perplexity
        ii. BLEU score
    c. Compare outputs generated using different prompts.

Text Book 2: Chapter 7:7.1 to 7.6

Module 5 :Transformers	**No. of Hrs: 8+4**

Transformers: Attention, Transformer Blocks, Parallelizing computation using a single matrix

X, The input: embeddings for token and position, The Language Modeling Head, More on Sampling, Training, Dealing with Scale, Interpreting the Transformer.

Laboratory Experiments :

1. Develop a python program to:
    a. Load a pretrained BERT model.
    b. Input a sentence.
    c. Extract and visualize attention weights.
    d. Interpret token relationships.
2. Write a Python program to:
    a. Load pretrained BERT Question Answering model.
    b. Provide a context paragraph.
    c. Accept user question.
    d. Extract answer from context.

Text Book 2: Chapter 8

Course Outcomes: At the end of the course, the student will be able to

1. **Describe** language structure, tokenization, subword models, edit distance, and basic statistical language models.
2. **Compute** probabilistic and vector-based representations, including N-grams, embeddings, similarity measures, and evaluation metrics.
3. **Illustrate** word embeddings and neural network classifiers, including Word2Vec, cosine similarity, and feedforward networks.
4. **Explain** Large Language Models and Transformer architectures, including model principles, architecture, text generation workflows, training, and evaluation.
5. **Apply** Large Language Models and Transformers to perform text generation, sequence modeling, embeddings, sampling, and evaluation tasks.

Textbooks:

1. Tanveer Siddiqui, U.S. Tiwary, "Natural Language Processing and Information Retrieval", Oxford University Press. 2008
2. Dan Jurafsky and James H. Martin, "Speech and Language Processing", 2024

Reference Books:

1. Akshay Kulkarni, Adarsha Shivananda, "Natural Language Processing Recipes - Unlocking Text Data with Machine Learning and Deep Learning using Python", Apress, 2019.
2. T V Geetha, "Understanding Natural Language Processing – Machine Learning and Deep Learning Perspectives", Pearson, 2024.
3. Gerald J. Kowalski and Mark.T. Maybury, "Information Storage and Retrieval systems", Kluwer Academic Publishers.

Web link

1. NLP basics and applications: https://www.youtube.com/watch?v=M7SWr5xObkA
2. Core NLP techniques : https://youtu.be/02QWRAhGc7g
3. NLP practical concepts: https://www.youtube.com/watch?v=CMrHM8a3hqw
4. NPTEL NLP course : https://onlinecourses.nptel.ac.in/noc23_cs45/preview
5. NPTEL NLP course : https://archive.nptel.ac.in/courses/106/106/106106211/

COs / POs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3	2	-	-	2	-	-	-	-	-	-	
CO2	3	3	1	2	3	-	-	-	-	-	-	
CO3	3	3	2	2	3	-	-	-	-	-	-	
CO4	3	2	1	2	3	-	-				-	
CO5	3	3	3	3	3	-	-				-	

Introduction to NLP

- **What is Natural Language Processing (NLP)**
- Language is the primary means of communication used by humans and tool to express the greater part of our ideas and emotions. It shapes thought and has a structure, and carries meaning. To express a thought, content helps represent the language in real-time.
- NLP is concerned with ***development of computational models of aspects of human language processing, there are two main reasons:***
  - To develop automated tools for language processing
  - To gain a better understanding of human communication
- Building computational models with human language-processing abilities requires a knowledge of how humans acquire, store, and process language.
- Historically, there have been **two major approaches to NLP:**
  - Rationalist approach
  - Empiricist approach

Introduction to NLP

- ***Rationalist Approach:*** Early approach, assumes the existence of some language faculty in the human brain. Supporters of this approach argue that it is not possible to learn a complex thing like natural language from limited sensory inputs.
- ***Empiricist approach:*** Do not believe in existence of a language faculty. Believe in the existence of some general organization principles such as pattern recognition, generalization, and association Learning of detailed structures takes place through the application of these principles on sensory inputs available to the child.

Origins of NLP

- The NLP includes speech processing and sometimes mistakenly termed natural language understanding-originated from machine translation research.
- Natural language processing includes both understanding (interpretation) and generation (production).
- We are concerned with text processing only - The area of *computational linguistics and its application.*

Linguistics - the scientific study of language

Linguists - a person who studies languages or linguistics.

Origins of NLP

- **Computational linguistics:** is similar to theoretical- and psycho-linguistics, but uses different tools.
  - **Theoretical linguistics** is more about the structural rules of language,
  - **psycho-linguistics** focuses on how language is used and processed in the mind.
- Theoretical linguistics explores the abstract rules and structures that govern language. It investigates universal **grammar, syntax, semantics, phonology, and morphology**.
- **Linguists** create models to explain how languages are structured and how meaning is encoded.
  - Eg. Most languages have constructs like noun and verb phrases.
- Theoretical linguists identify rules that describe and restrict the structure of languages (grammar).
- Psycho-linguistics focuses on the psychological and cognitive processes involved in language use.
- It examines how individuals acquire, process, and produce language.
- Researchers study language development in children and how the brain processes language in real-time.
  - Eg. Studying how children acquire language, such as learning to form questions ("What's that?").

Origins of NLP

- **Computational Linguistics Models:**
- Computational linguistics is concerned with the study of language using computational models of linguistic phenomena.
- It deals with the application of linguistic theories and computational techniques for NLP.
- In computational linguistics,
  - representing a language is a major problem;
  - Most knowledge representations tackle only a small part of knowledge.
- Representing the whole body of knowledge is almost impossible

Origins of NLP

- Computational models may be broadly classified under **knowledge-driven and data-driven categories**.
- **Knowledge-driven systems** rely on explicitly coded linguistic knowledge, often expressed as a set of handcrafted grammar rules.
  - Acquiring and encoding such knowledge is difficult and is the main bottleneck in the development of such systems.
- **Data-driven approaches** presume the existence of a large amount of data and usually employ some machine learning technique to learn syntactic patterns.
  - Performance of these systems is dependent on the quantity of the data and usually adaptive to noisy data.
- Main objective of the models is to achieve a balance between semantic (knowledge-driven) and data-driven approaches on one hand, and between theory and practice on the other.

Language & Knowledge

- Language & Knowledge
- Language is the medium of expression in which knowledge is deciphered.
- We are here considering the *text form of the language and the content of it as knowledge.*
- Language, being a medium of expression, is the outer form of the content it expresses.
- The same content can be expressed in different languages.
- Hence, to process a language means to process the content of it.
- As computers are not able to understand natural language, methods are developed to map its content in a formal language.
- The language and speech community considers a language as a set of sounds that, through combinations, conveys meaning to a listener.
- However, we are concerned with **representing and processing text only**.
- **Language (text) processing has different levels, each involving different types of knowledge.**

Language (text) processing has different levels, each involving different types of knowledge.

lexical analysis

- **lexical analysis**
  - Analysis of words.
  - Word-level processing requires morphological knowledge, i.e., knowledge about the structure and formation of words from basic units (morphemes).
  - The rules for forming words from morphemes are language specific.

Word vs Morpheme (Lexical Analysis)

- A **word** is a complete, independent unit of language that can stand alone, while a **morpheme** is the smallest meaningful building block that makes up a word. A single word can be made of one or multiple morphemes.
- **Key Differences**
- **Definition**:
- A **word** functions on its own in a sentence;
- a **morpheme** is the core, minimum unit of meaning or grammar.
- **Independence**: **Words** always stand alone, but **morphemes** can be *free* (standing alone as a word, like "cat") or *bound* (needing to attach to other parts, like the plural "-s").
- **Structure**: A **word** can be broken down into morphemes, but a **morpheme** cannot be broken down further without losing its specific meaning.
- **Examples of Breakdown**
- **One morpheme, one word**: *"dog"* = 1 word and 1 free morpheme.
- **Two morphemes, one word**: *"cats"* = 1 word made of 2 morphemes (*cat* + *-s*).
- **Three morphemes, one word**: *"unbreakable"* = 1 word made of 3 morphemes (*un-* + *break* + *-able*).

Syntactic analysis

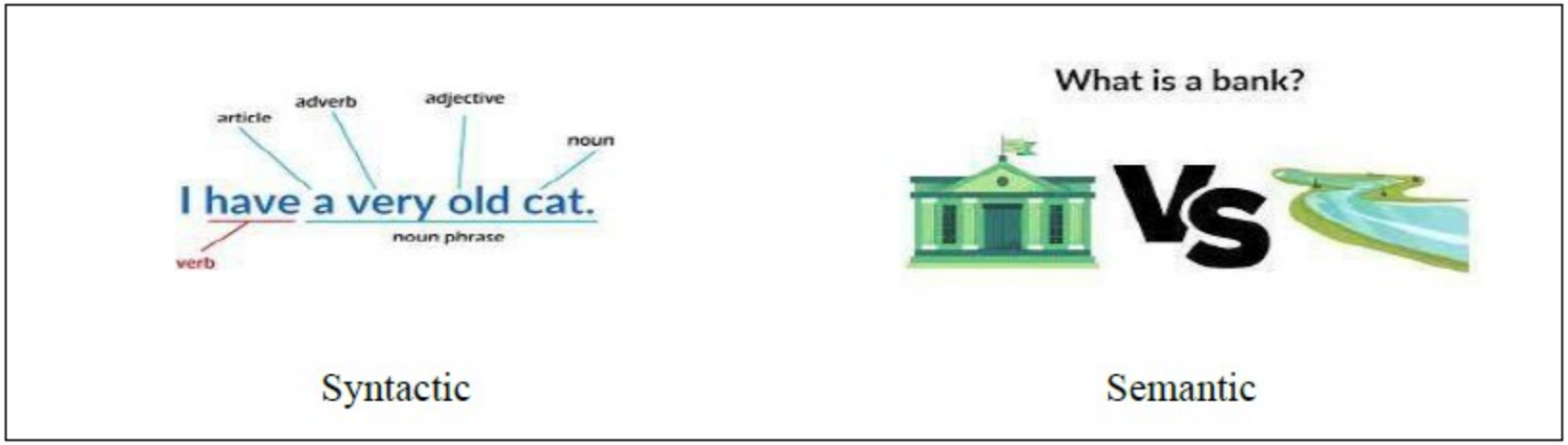
- **Syntactic analysis**
  - Considers a sequence of words as a unit, usually a sentence, and finds its structure.
  - Decomposes a sentence into its constituents (or words) and identifies how they relate to each other.
  - It captures grammaticality or non-grammaticality of sentences by looking at constraints like word order, number, and case agreement.
  - This level of processing requires ***syntactic knowledge (How words are combined to form larger units such as phrases and sentences)***
  - For example:
  - 'I went to the market' is a valid sentence whereas 'went the I market to' is not.
  - 'She is going to the market' is valid, but 'She are going to the market' is not.

Semantic analysis

- It is associated with the meaning of the language.
- Semantic analysis is concerned with creating meaningful representation of linguistic inputs.
- Eg. 'Colorless green ideas sleep furiously' - syntactically correct, but semantically anomalous

Semantic analysis

- A word can have a number of possible meanings associated with it. But in a given context, only one of these meanings participates.



Semantic analysis

- Finding out the correct meaning of a particular use of word is necessary to find meaning of larger units.
- Eg.
  - Kabir and Ayan are married.
  - *Kabir and Suha are married.*
- Syntactic structure and compositional semantics fail to explain these interpretations.
- This means that *semantic analysis requires pragmatic knowledge besides semantic and syntactic knowledge.*
- **Pragmatics helps us understand how meaning is influenced by context, social factors, and speaker intentions.**

Pragmatic analysis

- **Pragmatic analysis**
- The highest level of processing, deals with the purposeful use of sentences in situations.
- It requires knowledge of the world, i.e., knowledge that extends beyond the contents of the text.
- **Example 1: Request vs. Question**

  Sentence:
  - **Can you open the window?**
  - **Literal (Semantic) Meaning**
  - The speaker is asking:
  - **"Do you have the ability to open the window?"**
  - **Pragmatic Meaning**
  - The speaker actually means:
  - **"Please open the window."**
  - Here, the computer must understand that it is a **polite request**, not a question about ability.
- **Example 2: World Knowledge**

  Sentence:
  - **The baby is crying.**
  - Literal meaning:
  - The baby is crying.
  - Pragmatic understanding:
  - The baby may be hungry.
  - The baby may be sleepy.
  - The baby may need attention.
  - This requires **world knowledge**, which is **not explicitly stated** in the sentence.

Discourse Analysis

- **Discourse Analysis**
- Attempts to interpret the structure and meaning of even larger units, e.g., at the paragraph and document level, in terms of words, phrases, clusters, and sentences.
- It requires the resolution of anaphoric references and identification of discourse structure.



Anamorphic Reference

Discourse Analysis

- Pragmatic knowledge may be needed for resolving anaphoric references.
- Example:
  - The district administration refused to give the trade union permission for the meeting because they feared violence.
    - **They refers ?**
  - The district administration refused to give the trade union permission for the meeting because they oppose government.
    - **They refers ?**
- For example, in the above sentences, resolving the anaphoric reference 'they' requires pragmatic knowledge.

The Challenges of NLP

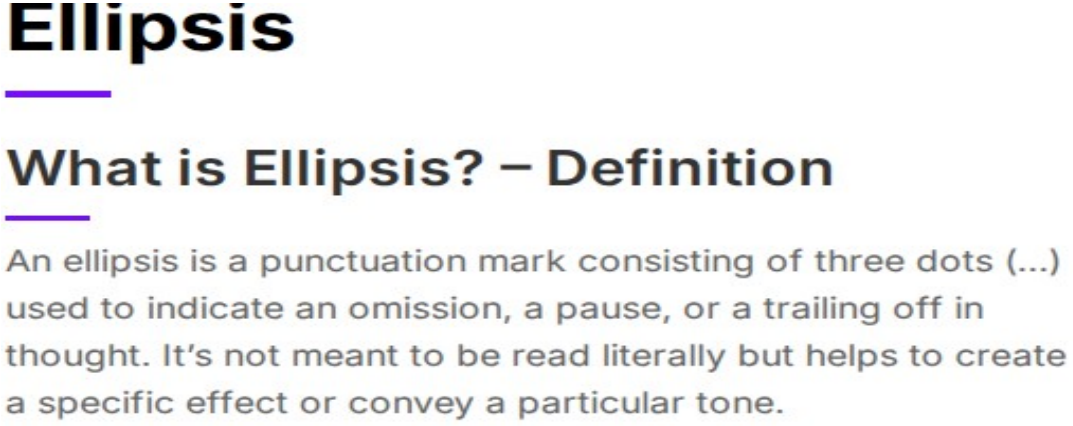
- Natural languages are highly ambiguous and vague, achieving precise representation of content can be difficult.
- The inability to capture all the required knowledge.
- Identifying its semantics.
- A language keeps on evolving. New words are added continually and existing words are introduced in new context. (eg. 9/11 - terrorist act on WTC)

- Solution: The only way machines can learn is by considering its context, context of a word is defined by co-occurring words.
- The frequency of a word being used in a particular sense also affects its meaning.

The Challenges of NLP

- **Idioms, metaphor, and ellipses** add *more complexity to identify the meaning of the written text.*
- *Example: "The old man finally kicked the bucket"*
- ***"kicked the bucket" is a well-known* Idiom, meaning is to "*to die.*"**
-
- ***"Time is a thief." Metaphor suggests "time robs you of valuable moments or experiences in life".***

- "I'm going to the store, and you're going to the party, right?"
- "Yes, I am…"
- **Ellipses refer to the omission of words or phrases in a sentence. (represented by "…")**



idioms

Idioms are phrases or expressions where the meaning cannot be guessed by looking at the individual words. Instead, the words work together to create a completely new, figurative meaning that must be learned as a whole unit. Merriam-Webster +1

Common Examples

- **A piece of cake:** Something very easy to do.
- **Break the ice:** To start a conversation in a new or tense situation.
- **Cost an arm and a leg:** To be very expensive.
- **Spill the beans:** To reveal a secret.
- **Under the weather:** Feeling slightly sick or unwell. EF +3

metaphor

/ˈmɛtəfə/

A metaphor is **a figure of speech that compares two different things by saying one thing is another**. It does not use the words "like" or "as," which separates it from a simile. Instead, it makes a direct claim to show a hidden similarity. Grammarly +1



Common Examples

- "Time is a thief." (Time steals our moments.)
- "The classroom was a zoo." (The room was loud and wild.)
- "He has a heart of gold." (He is very kind.)
- "She is a shining star." (She is bright and talented.) YouTube · Learn and Play ...

The Challenges of NLP

- The **ambiguity of natural languages is another difficulty (explicit as well as implicit sources of knowledge).**
- **Word Ambiguity: Example: 'Taj' - a monument, a brand of tea, or a hotel.**
- **“Can” – ambiguous in its part-of-speech. ('Part-of-speech tagging' algorithm) (I can Swim & Open the can)**
- “Bank” is ambiguous in its meaning. ('word sense disambiguation' algorithm) o Structural ambiguity - A sentence may be ambiguous
- 'Stolen rifle found by tree.'
- Structural Ambiguity (Syntactic Ambiguity)
- (**I saw the man with a telescope.**
  - Possible meanings:
  - I used a telescope to see the man.
  - The man was carrying a telescope.
- Verb sub-categorization may help to resolve
- Probabilistic parsing - statistical models to predict the most likely syntactic structure.

Structural Ambiguity

Example: “Stolen rifle found by tree.”

It has two possible meanings:

Rifle was found near a tree.

Rifle was found by someone near the tree.

The ambiguity occurs because **“by tree” can be interpreted differently based on sentence structure**.

- • A number of grammars have been proposed to describe the structure of sentences.
- It is almost impossible for grammar to capture the structure of all and only meaningful text.

- **a) Verb Sub-categorization**
- Knowledge about which sentence patterns a verb usually follows helps reduce ambiguity.
- **Example:**
- **Give** usually requires:
  - Subject + Verb + Object + Indirect Object
  - *John gave Mary a book.*
- This knowledge helps the parser build the correct structure.

- **Probabilistic Parsing**
- Instead of considering all possible sentence structures equally, **probabilistic parsers** use **statistical models** trained on large text corpora to choose the **most likely syntactic structure**.
- For example, in:
- **I saw the man with a telescope.**
- A probabilistic parser selects the interpretation that is most common in real language usage.

Grammar Limitations

- Many grammars have been proposed to describe sentence structure, such as:
  - Context-Free Grammar (CFG)
  - Dependency Grammar
  - Phrase Structure Grammar
- **Challenge**
- It is **almost impossible** for a grammar to describe **all and only meaningful sentences** because:
  - Natural language is highly flexible.
  - New words and expressions appear frequently.
  - People often use incomplete or informal sentences.
  - Different contexts can change the meaning of the same sentence.
- **Example**
- **Grammatically correct but meaningless:**
- **Colorless green ideas sleep furiously.**
  - Grammar: ✔ Correct
  - Meaning: ✖ Nonsensical
- **Grammatically incorrect but meaningful:**
- **Me go home now.**
  - Grammar: ✖ Incorrect
  - Meaning: ✔ Easily understood
- This demonstrates that **grammar alone cannot capture the full meaning of natural language**. NLP systems must also use **semantics (meaning), pragmatics (context), and world knowledge**.

Many grammars have been proposed to describe sentence structure, such
Context-Free Grammar (CFG)
Dependency Grammar
Phrase Structure Grammar

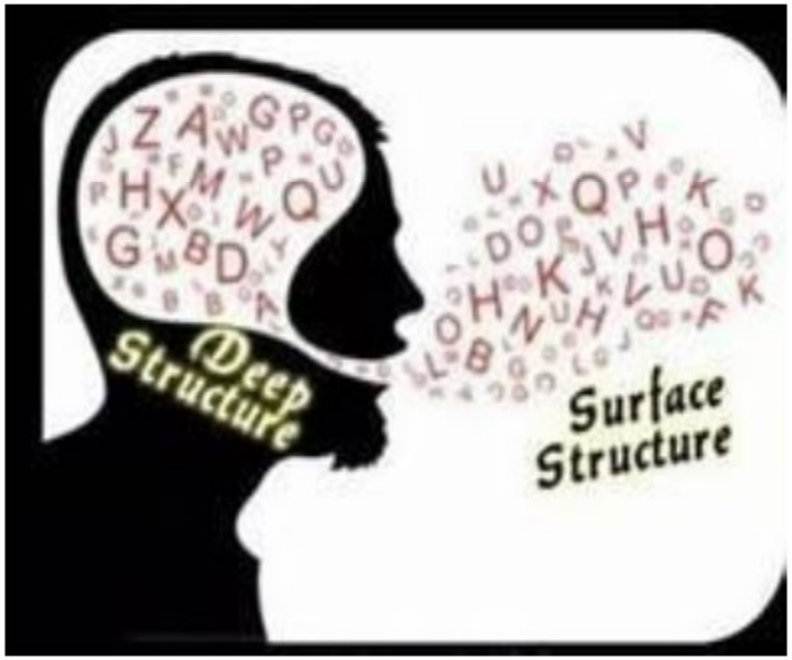
Grammar	Main idea	Focus	Simple example
Context-Free Grammar (CFG)	A sentence can be generated using grammatical rules	Rules for combining **categories**	S → NP + VP
Dependency Grammar	Words depend on other words	**Relationships between individual words**	*eats* → subject: *John*, object: *apple*
Phrase Structure Grammar	A sentence is divided into nested phrases	**Constituents / phrases**	Sentence → Noun Phrase + Verb Phrase

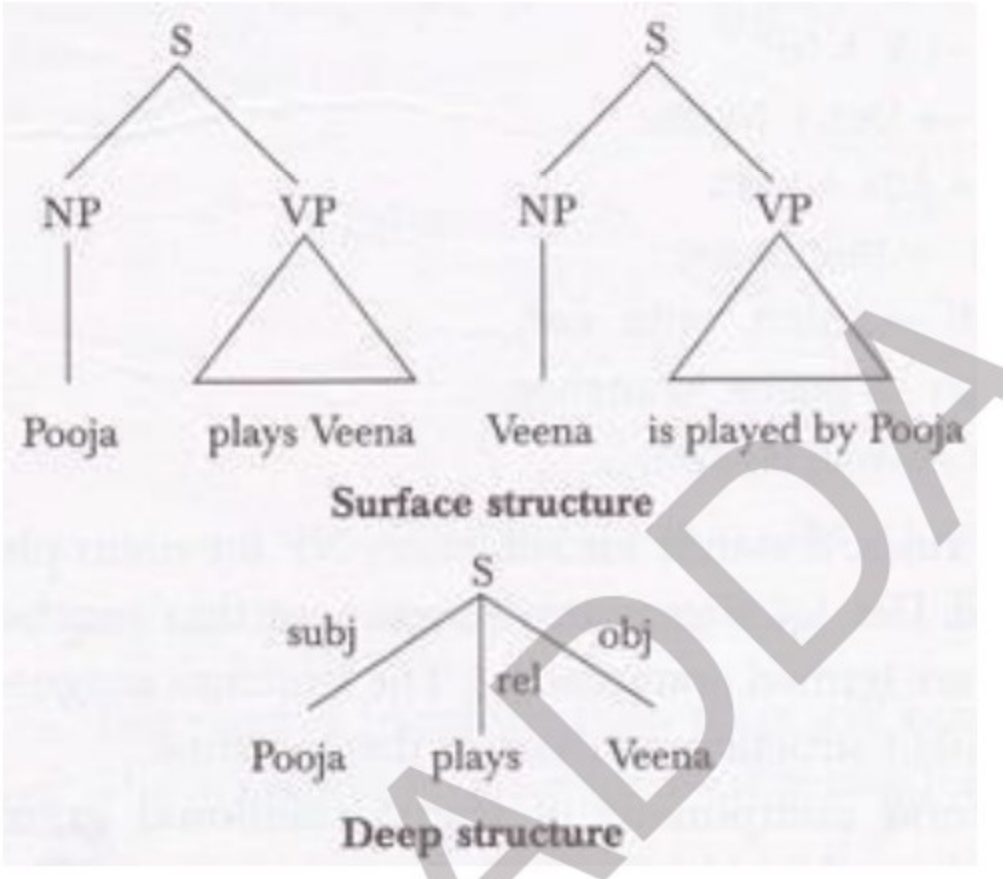
Language and Grammar

- Language Grammar: Grammar defines language and consists of rules that allow parsing and generation of sentences, serving as a foundation for natural language processing.
- Syntax vs. Semantics: Although syntax and semantics are closely related, a separation is made in processing due to the complexity of world knowledge influencing both language structure and meaning.
- Challenges in Language Specification: Natural languages constantly evolve, and the numerous exceptions make language specification challenging for computers.
- Different Grammar Frameworks: Various grammar frameworks have been developed, including transformational grammar, lexical functional grammar, and dependency grammar, each focusing on different aspects of language such as derivation or relationships.
- Chomsky's Contribution: Noam Chomsky's generative grammar framework, which uses rules to specify grammatically correct sentences, has been fundamental in the development of formal grammar hierarchies.

Language and Grammar

- Chomsky argued that phrase structure grammars are insufficient for natural language and proposed *transformational grammar in Syntactic Structures (1957). He suggested that each sentence has two levels:* ***a deep structure and a surface structure (as shown in Fig 1), with transformations mapping one to the***





Language and Grammar

- Chomsky argued that an utterance is the surface representation of a 'deeper structure' representing its meaning.
- The deep structure can be transformed in a number of ways to yield many different surface-level representations.
- Sentences with different surface-level representations having the same meaning, share a common deep-level representation.

Pooja plays veena.
Veena is played by Pooja.

Language and Grammar

- Both sentences have the same meaning, despite having different surface structures (roles of subject and object are inverted).
- Transformational grammar has three components:
  - Phrase structure grammar: Defines the basic *syntactic structure of sentences.*
  - Transformational rules: Describe how *deep structures can be transformed into different surface structures.*
  - Morphophonemic rules: Govern the relationship structure of a sentence (its syntax) influences the form of the words in terms of *sound and pronunciation (phonology).*

Language and Grammar

- Phrase structure grammar: Defines the basic *syntactic structure of sentences – Example set of Rules*
- *Sentences that can be generated using these rules are termed as grammatical*

S ➔ NP + VP VP ➔ V + NP NP ➔ Det + Noun V ➔ Aux + Verb	Det ➔ the, a, an, ... Verb ➔ catch, write, eat, ... Noun ➔ police, snatcher, ... Aux ➔ will, is, can, ...

S : sentence	Vt : transitive verb
NP : noun phrase	Vi : intransitive verb
VP : verb phrase	VL : linking verb
N : noun	Comp : complement
V : verb	Prep : preposition
D or Det : determiner	Pres : present
Pron : pronoun	PP : prepositional phrase
Prop N : proper name	M : modal
Pred : predicate	Be : the verb "be"
M : modal	Adv. P : adverb phrase
Be : the verb "be"	Adj. P : adjective phrase

Transformational rules:

- Transformational rules: Describe how *deep structures can be transformed into different surface structures.*
- ***Transformation rules, transform one phrase-maker (underlying) into another phrase-marker (derived). These rules are applied on the terminal string generated by phrase structure rules. It transforms one surface representation into another, e.g., an active sentence into passive one.***
- Consider the active sentence: "*The police will catch the snatcher.*"

[NP1 - Aux - V - NP2] ❼ [NP2 - Aux + be + en - V - by + NP1]

Application of Phase Structure Rule to this sentence

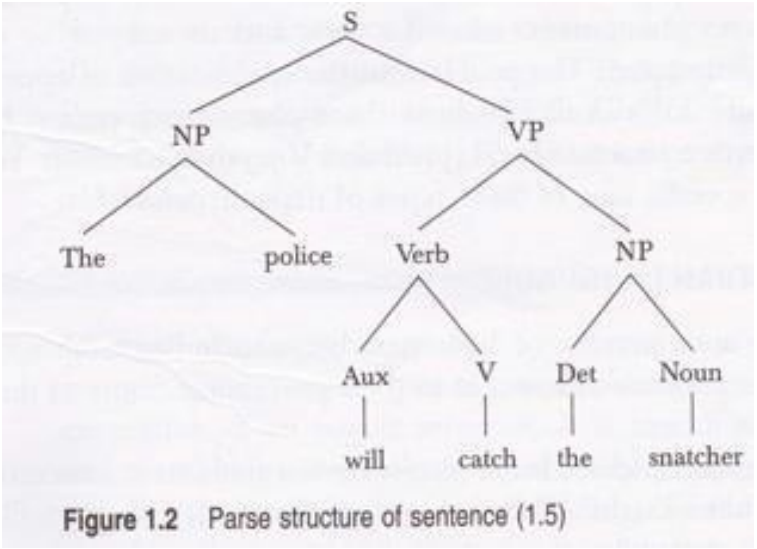


Figure 1.2 Parse structure of sentence (1.5)

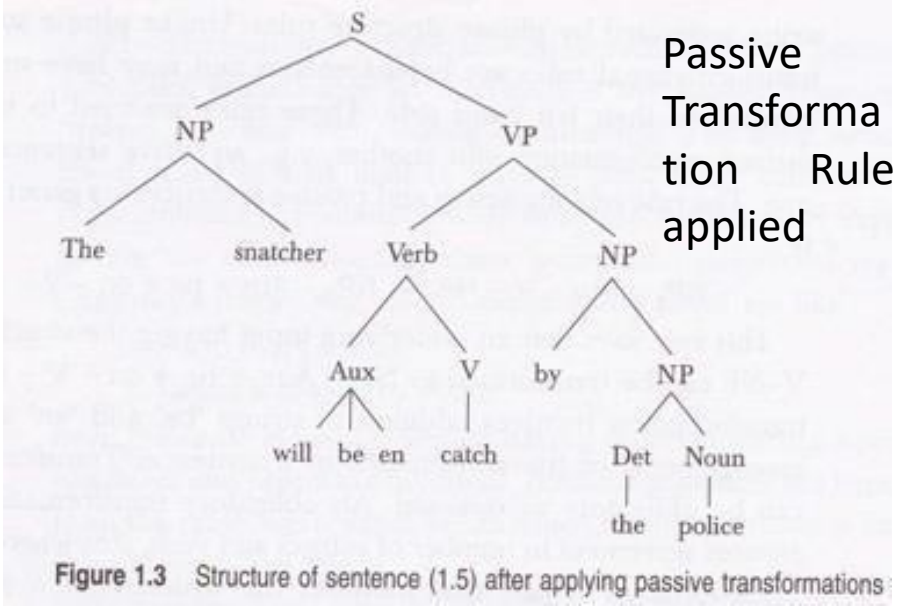


Figure 1.3 Structure of sentence (1.5) after applying passive transformations

Passive Transformation Rule applied

Morphophonemic Rule

- **Phrase structure (b) Passive Transformation**
- The passive transformation rules will convert the sentence into
- The + culprit + will + be + en + catch + by + police
- ***Morphophonemic Rule: Another transformational rule will then reorder 'en + catch' to 'catch + en' and subsequently one of the morphophonemic rules will convert 'catch + en' to 'caught'.***

NLP Applications

- **Machine Translation**
- This refers to automatic translation of text from one human language to another. In order to carry out this translation, it is necessary to have an understanding of words and phrases, grammars of the two languages involved, semantics of the languages, and word knowledge.
- **Speech Recognition**
- This is the process of mapping acoustic speech signals to a set of words. The difficulties arise due to wide variations in the pronunciation of words, homonym (e.g. dear and deer) and acoustic ambiguities (e.g., in the rest and interest).

NLP Applications

- **Speech Synthesis**
- Speech synthesis refers to automatic production of speech (utterance of natural language sentences). Such systems can read out your mails on telephone, or even read out a storybook for you.
- **Information Retrieval**
- This focuses on identifying relevant documents for a user's query using NLP techniques in information retrieval. Methods like indexing, word sense disambiguation, query modification, and knowledge bases improve IR performance, including query expansion. Lexical resources like WordNet, LDOCE, and Roget's Thesaurus enhance these systems. These tools help refine search results and improve accuracy.

NLP Applications

- **Information Extraction**
- An information extraction system captures and outputs factual information contained within a document. Query is specified as pre-defined templates. System identifies a subset of information within a document that fits the pre-defined template.
- **Question Answering**
- Given a question and a set of documents, a question answering system attempts to find the precise answer, or at least the precise portion of text in which the answer appears. A question answering system requires more NLP than an information retrieval system or an information extraction system. It requires not only precise analysis of questions and portions of texts but also semantic as well as background knowledge to answer certain type of questions.
- **Text Summarization**
- This deals with the creation of summaries of documents and involves syntactic, semantic, and discourse level processing of text.

Words and Tokens

Words

How many words in a sentence?

```
They picnicked by the pool, then
lay back on the grass and looked at
the stars.
```

16 words

- if we don't count punctuation marks as words

18 if we count punctuation

How many words in an utterance?

"I do uh main- mainly business data processing"

Disfluencies

- Fragments *main-*
- *Filled pauses: uh* and *um*

- Should we consider these to be words?

How many words in a sentence?

`They picnicked by the pool, then lay back on the grass and looked at the stars.`

Type: an element of the vocabulary V

- The number of types is the vocabulary size |V|

Instance: an instance of that type in running text.

- 14 types and 16 instances (if we ignore punctuation).

More questions: Are `They` and `they` the same word?

How many words in a sentence?

I'm

Orthographically one word (in the English writing system)

But **grammatically** two words:

1. the subject pronoun I
2. the verb 'm, short for am

How many words in a sentence?

Not every written language uses spaces!!

Chinese, Japanese and Thai don't!

How to choose tokens in Chinese

Chinese words are composed of characters called "**hanzi" (汉字, hànzì)** (or sometimes just "**zi**")

Each one represents a meaning unit called a morpheme.

Each word has on average 2.4 of them.

But deciding what counts as a word is complex and not agreed upon.

How to choose tokens in Chinese?

姚明进入总决赛 “Yao Ming reaches the finals”

- yáo míng jìn rù zǒng jué sài

3 words?

姚明 进入 总决赛

YaoMing reaches finals

Chinese Treebank

5 words?

姚 明 进入 总 决赛

Yao Ming reaches overall finals

Peking University

7 words?

姚 明 进 入 总 决 赛

Yao Ming enter enter overall decision game

Just use characters

Tokenization across languages

- So in Chinese we use characters (zi) as tokens
  - But that doesn't work for, e.g., Thai and Japanese
  - These differences make it hard to use words as tokens
- And there's another reason why we don't use words as tokens!

There are simply too many words!

	Types = |V|	Instances = N
Shakespeare	31 thousand	884,000
Brown Corpus	38 thousand	1 million
Switchboard conversations	20 thousand	2.4 million
COCA	2 million	440 million
Google N-grams	13+ million	1 trillion

The bigger the corpus, the more word types!

There are simply too many words!

N = number of instances

$|V|$ = number of types in vocabulary V

Heaps Law = Herdan's Law

$$|V| = kN^{\beta}$$ ← Roughly 0.5

Vocab size for a text goes up with the square root of its length in words

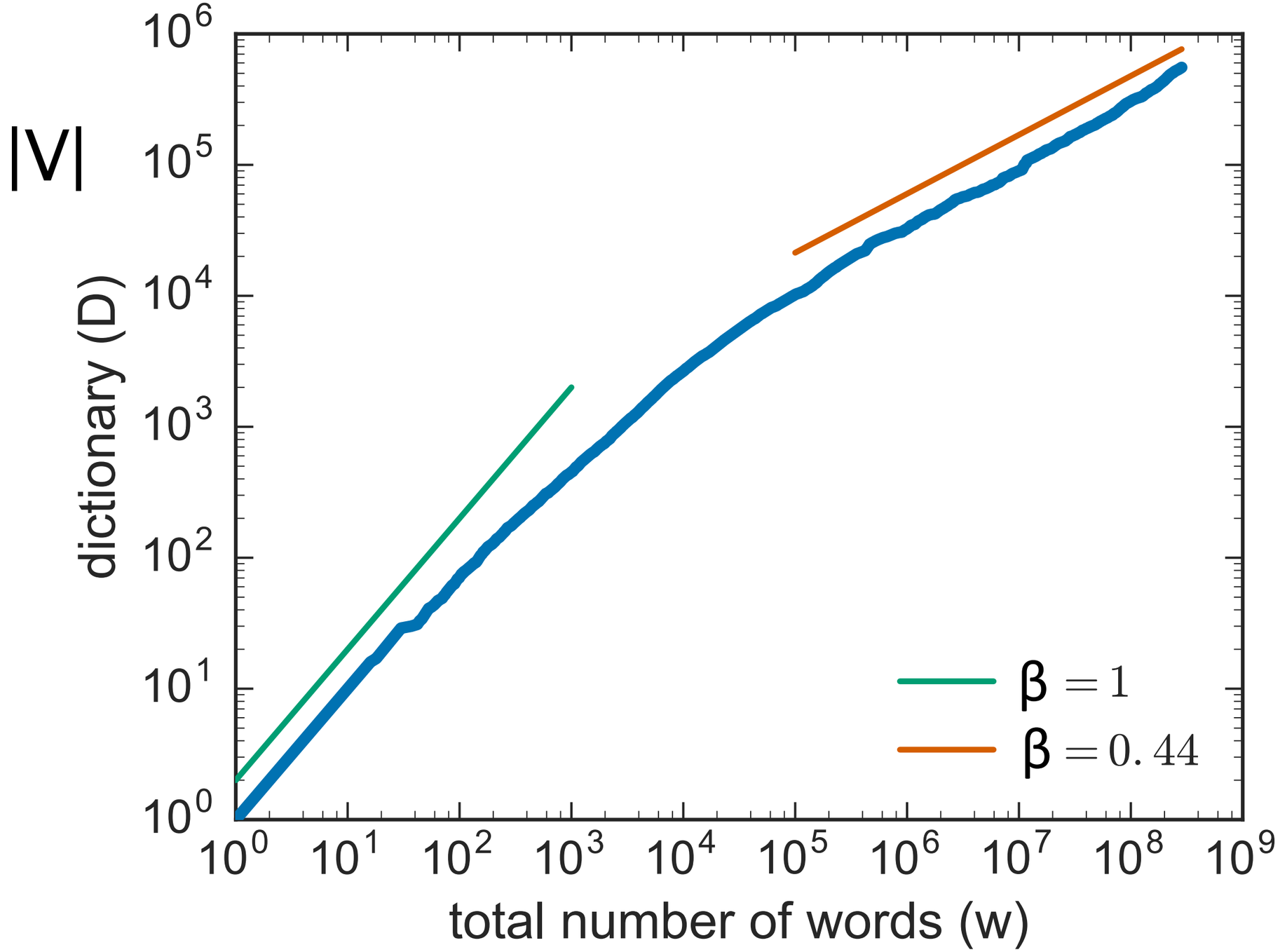
Two kinds of words

Function words

- of, the, is, and, una, 是,

Content words

- mango, braise, snowy, feliz, 北京



Tria, Loreto, Servedio, 2018

Why is too many words a problem?

No matter how big our vocabulary

There will always be words we missed!

We will always have unknown words!

Words and Subwords

Because of these three problems:

1. Many languages don't have orthographic words
2. Defining words post-hoc is challenging
3. The number of words grows without bound

NLP systems don't use words, but smaller **subwords**

Words and Tokens

Words

Words and Tokens

Morphemes

Words have parts

Morpheme: a minimal meaning-bearing unit in a language.

`fox`: one morpheme

`cats`: two morphemes `cat` and `-s`

Morphology: the study of morphemes

Morphemes in English and Chinese

Doc work-ed care-ful-ly wash-ing the glass-es

梅　干 菜　　用 清　水　泡　软 ，捞　　出 后 ，沥　干
plum dry vegetable use clear water soak soft , remove out after , drip dry
切　碎
chop fragment

Soak the preserved vegetable in water until soft, remove, drain, and chop

Types of morphemes

root: central morpheme of the word
- supplying the main meaning

affix: adding additional meanings

worked

root work

affix -ed

glasses

root glass

affix -es

Types of affixes

Inflectional morphemes

- grammatical morphemes
- often syntactic role like agreement
  - `-ed` past tense on verbs
  - `-s/-es` plural on nouns

Derivational morphemes

- more idiosyncratic in application and meaning
- often change grammatical class
  - *`care`* (noun)
    - + *`-full`* → `careful` (*adjective*)
    - + *`-ly`* → *`carefully`* (*adverb*)

Clitics

A morpheme that acts syntactically like a word but:

- is reduced in form
- and attached to another word

English: `'ve` in `I've` (`'ve` can't appear alone)

English: `’s` in `the teacher’s book`

French: `l’` in `l’opera`

Arabic: `b` ‘by/with’, `w` ‘and’.

Morphological Typology

Dimensions along which languages vary

Two are salient for tokenization:

1. number of morphemes per word
2. how easy it is to segment the morphemes

Number of morphemes per word

Few. *Cantonese, spoken in Guangdong, Guangxi, Hong Kong*

keoi5 waa6 cyun4 gwok3 zeoi3 daai6 gaan1 uk1 hai6 ni1 gaan1

he say entire country most big building house is this building

“He said the biggest house in the country was this one”

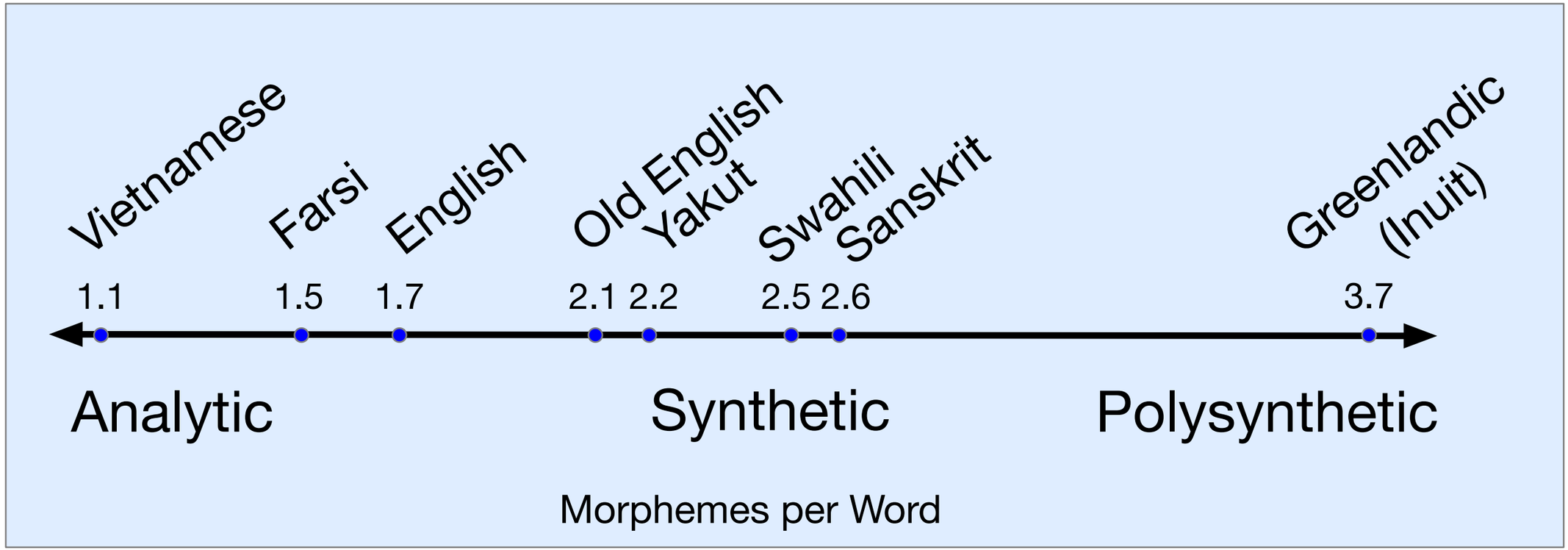
Many. Koryak, Kamchatka peninsula in Russia,

t-ə-nkʼe-mejŋ-ə-jetemə-nni-k

1SG.S-E-midnight-big-E-yurt.cover-E-sew-1SG.S[PFV]

“I sewed a lot of yurt covers in the middle of a night.”

Joseph Greenberg (1960) scale



How easily segmentable

Agglutinative languages like Turkish
- Very clean boundaries between morphemes

Fusion languages
- a single affix may conflate multiple morphemes,

- Russian `-om` in `stolom` (table-SG-INSTR- DECL1)
  - instrumental, singular, and first declension.
- English `-s` in "`She reads the article`"
  - Means both "third person" and "present tense"

These are tendencies rather than absolutes

Words and Tokens

Morphemes

Words and Tokens

Unicode

Unicode

a method for representing text written using

- any character (more than 150,000!)
- in any script (168 to date!)
- of the languages of the world
  - Chinese, Arabic, Hindi, Cherokee, Ethiopic, Khmer, N'Ko,…
  - dead ones like Sumerian cuneiform
  - invented ones like Klingon
  - plus emojis, currency symbols, etc.

ASCII: Some history for English

1960s American Standard Code for Information Interchange

- 1 byte per character
  - In principle 256 characters
  - But high bit set to 0
  - So 7 bits = 128
  - However only 95 used
- The rest were for teletypes



ASCII: Some history for English

Ch	Hex	Dec	Ch	Hex	Dec		Ch	Hex	Dec	Ch	Hex	Dec
<	3C	60	@	40	64	...	\	5C	92	`	60	96
=	3D	61	A	41	65	...	[	5D	93	a	61	97
>	3E	62	B	42	66	...	^	5E	94	b	62	98
?	3F	63	C	43	67	...	_	5F	95	c	63	99

```
h   e   l   l   o
68  65  6C  6C  6F
```

ASCII wasn't enough!

Spanish: Señor- respondió Sancho

This sentence has non-ASCII ñ and ó

About 100,000 **Chinese/CJKV** characters (Chinese, Japanese, Korean, or Vietnamese)

Devanagari script for 120 languages like **Hindi**, Marathi, Nepali, Sindhi, Sanskrit, etc.

अनुच्छेद १(एक): सभी मनुष्य जन्म से स्वतन्त्र तथा मर्यादा और अधिकारों में समान होते हैं। वे तर्क और विवेक से सम्पन्न हैं तथा उन्हें भ्रातृत्व की भावना से परस्पर के प्रति कार्य करना चाहिए।

Code Points

Unicode assigns a unique ID, a **code point,** to each of its 150,000 characters

1.1 million possible code points

- 0 – 0x10FFFF

Written in hex, with prefix "U+"

- a is U+0061 which = 0x0061

First 127 code points = ASCII

- For backwards compatibility

Some code points

```
0061   a   LATIN SMALL LETTER A
0062   b   LATIN SMALL LETTER B
0063   c   LATIN SMALL LETTER C
00F9   ù   LATIN SMALL LETTER U WITH GRAVE
00FA   ú   LATIN SMALL LETTER U WITH ACUTE
00FB   û   LATIN SMALL LETTER U WITH CIRCUMFLEX
00FC   ü   LATIN SMALL LETTER U WITH DIAERESIS
8FDB   进
8FDC   远
8FDD   违
8FDE   连
1F600  😀  GRINNING FACE
1F00E  🀎  MAHJONG TILE EIGHT OF CHARACTERS
```

A code point has no visuals; it is **not** a glyph!

Glyphs are stored in **fonts**: **a** or *a* or a or a

But one code point (U+0061, abstract "LATIN SMALL A") represents all those different a's!

Encodings and UTF-8

We don't stick code points directly in files

We store **encodings** of chars.

The most popular encoding is UTF-8

Most of the web is stored in UTF-8

Encodings

`hello` has these 5 code points:

U+0068 U+0065 U+006C U+006C U+006F

How to write in a file?

There are more than 1 million code points

So would need 4 bytes (or 3 but 3 is inconvenient):

00 00 00 68 00 00 00 65 00 00 00 6C 00 00 00 6C 00 00 00 6F

But that would make files very long!

- Also zeros are bad (since mean "end of string" in ASCII)

Instead: Variable Length Encoding

UTF-8 (Unicode Transformation Format 8)

For the first 127 code points, same as ASCII

UTF-8 encoding of `hello` is :

- 68 65 6C 6C 6F

Code points ≥128 are encoded as a sequence of 2, 3, or 4 bytes

- In range 128 - 255, so won't be confused with ASCII
- First few bits say if its 2-byte, 3-byte, or 4-byte

UTF-8 Encoding

Code Points		UTF-8 Encoding			
From - To	**Bit Value**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**
U+0000-U+007F	0xxxxxxx	xxxxxxxx			
U+0080-U+07FF	00000yyy yyxxxxxx	110yyyyy	10xxxxxx		
U+0800-U+FFFF	zzzzyyyy yyxxxxxx	1110zzzz	10yyyyyy	10xxxxxx	
U+010000-U+10FFFF	000uuuuu zzzzyyyy yyxxxxxx	11110uuu	10uuzzzz	10yyyyyy	10xxxxxx

yyy yyxxxxxx

ñ, code point U+00F1, = 00000**000 11110001**

- Gets encoded with pattern 110yyyyy 10xxxxxx
- So is mapped to a two-byte bit sequence
- 110**00011** 10**110001** = 0xC3B1.

UTF-8 encoding

The first 127 characters (ASCII) map to 1 byte

Most remaining characters in European, Middle Eastern, and African scripts map to 2 bytes

Most Chinese, Japanese, and Korean characters map to 3 bytes

Rarer CJKV characters, emojis/symbols map to 4 bytes.

UTF-8 encoding

Efficient: fewer bytes for common characters,

Doesn't use **zero bytes** (except for NULL character U+0000),

Backwards compatible with **ASCII**,

Self-synchronizing,

- If a file is corrupted, the nearest character boundary is always findable by moving only up to 3 bytes

UTF-8 and Python 3

Python 3 strings stored internally as Unicode

- each string a sequence of Unicode code points
- string functions, regex apply natively to code points.
  - **len() returns string length in code points, not bytes**

Files need to be encoded/decoded when written or read

- Every file is stored in some encoding
- ***No such thing as a text file without an encoding***
  - If it's not UTF-8 it's something older like ASCII or iso_8859_1

Words and Tokens

Unicode

Words and Tokens

Byte Pair Encoding

The NLP standard for tokenization

Instead of

- white-space / orthographic words
  - Lots of languages don't have them
  - The number of words grows without bound
- Unicode characters
  - Too small as tokens for many purposes
- morphemes
  - Very hard to define

We use the data to tell us how to tokenize.

Why tokenize?

Using a deterministic series of tokens means systems can be compared equally

- Systems agree on the length of a string

Algorithms like perplexity assume all texts have a fixed tokenization

Eliminates the problem of unknown words

If some word occurs in test set but not training set, we still know how to segment it into known tokens.

Subword tokenization

Two most common algorithms:

- **Byte-Pair Encoding (BPE)** (Sennrich et al., 2016)
- **Unigram language modeling tokenization** (Kudo, 2018) (sometimes confusingly called "SentencePiece" after the library it's in)

All have 2 parts:

- A token **learner** that takes a raw training corpus and induces a vocabulary (a set of tokens).
- A token **encoder/segmenter** that takes a raw test sentence and tokenizes it according to that vocabulary

Byte Pair Encoding (BPE) token learner

Iteratively merge frequent neighboring tokens to create longer tokens.

Repeat:

- Choose most frequent neighboring pair ('A', 'B')
- Add a new merged symbol ('AB') to the vocabulary
- Replace every 'A' 'B' in the corpus with 'AB'.

Until *k* merges

Vocabulary

[A, B, C, D, E]

[A, B, C, D, E, AB]

[A, B, C, D, E, AB, CAB]

Corpus

A B D C A B E C A B

AB D C AB E C AB

AB D CAB E CAB

BPE token learner algorithm

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

Byte Pair Encoding (BPE) Addendum

Generally run **within** space-separated words

Don't merge across word boundaries

- First separate corpus by whitespace or similar, using specialized regular expressions
- This gives a set of starting strings, with whitespace attached to start of each strong
- Counts come from the corpus, but can only merge within strings.

BPE token learner

Original (very fascinating🙄) corpus:

`set␣new␣new␣renew␣reset␣renew`

Put space token at start of words

corpus		vocabulary
2	␣ n e w	␣, e, n, r, s, t, w
2	␣ r e n e w	
1	s e t	
1	␣ r e s e t	

BPE token learner

corpus

2 ␣ n e w
2 ␣ r e n e w
1 s e t
1 ␣ r e s e t

vocabulary

␣, e, n, r, s, t, w

Merge n e to ne (count 4 = 2 new + 2 renew)

corpus

2 ␣ ne w
2 ␣ r e ne w
1 s e t
1 ␣ r e s e t

vocabulary

␣, e, n, r, s, t, w, ne

BPE token learner

corpus	vocabulary
2 ␣ ne w	␣, e, n, r, s, t, w, ne
2 ␣ r e ne w	
1 s e t	
1 ␣ r e s e t	

Merge ne w to new (count 4)

corpus	vocabulary
2 ␣ new	␣, e, n, r, s, t, w, ne, new
2 ␣ r e new	
1 s e t	
1 ␣ r e s e t	

BPE token learner

corpus	vocabulary
2 ␣ new	␣, e, n, r, s, t, w, ne, new
2 ␣ r e new	
1 s e t	
1 ␣ r e s e t	

Merge ␣ r to ␣r (count 3) and ␣r e to ␣re (count 3)

corpus	vocabulary
2 ␣ new	␣, e, n, r, s, t, w, ne, new, ␣r, ␣re
2 ␣re new	
1 s e t	
1 ␣re s e t	

System has learned prefix re- !

BPE

The next merges are:

merge	current vocabulary
(␣, new)	␣, e, n, r, s, t, w, ne, new, ␣r, ␣re, ␣new
(␣re, new)	␣, e, n, r, s, t, w, ne, new, ␣r, ␣re, ␣new, ␣renew
(s, e)	␣, e, n, r, s, t, w, ne, new, ␣r, ␣re, ␣new, ␣renew, se
(se, t)	␣, e, n, r, s, t, w, ne, new, ␣r, ␣re, ␣new, ␣renew, se, set

BPE **encoder** algorithm

Tokenize a test sentence: run each merge learned from the training data:

- Greedily, in the order we learned them
- (test frequencies don't play a role)

First: segment each test word into characters

Then run rules: (1) merge every n e to ne, (2) merge ne w to new, (3) `␣r,` (4) `␣re` etc.

Result:

- Recreates training set words
- But also learns subwords like `␣re` that might appear in new words like rearrange

BPE and Unicode

We run BPE on large Unicode corpora, with vocabulary sizes of 50,000 to 200,000

On individual bytes of UTF-8-encoded text

- Not on Unicode characters
- BPE rediscovers 2-byte and common 3-byte UTF-8 sequences
- Only 256 possible values of a byte, so no unknown tokens
- (BPE might learn a few illegal UTF-8 sequences across character boundaries, but these can be filtered)

Visualizing GPT4o tokens

Tat Dat Duong's Tiktokenizer visualizer



Tokens: 11865, 8923, 11, 31211, 6177, 23919, 885, 220, 19427, 7633, 18887, 147065, 0

Most words are tokens, w/initial space

Clitics like 's

- Are segmented off Jane
- But part of frequent words like she's

Numbers segmented into chunks of 3 digits

Anyhow and ·anyhow are segmented differently

Some of this is from preprocessing

- regular expressions for chunking digits, stripping clitics

Tokenizing across languages

- Even though BPE tokenizers are multilingual
- LLM training data is still vastly dominated by English
  - Most BPE tokens used for English, leaving less for other languages
  - Words in other languages are often split up

Tokenization is better in English

Tat Dat Duong's Tiktokenizer visualizer on GPT4o

A recipe sentence in two languages

English: 18 tokens; no words are split into multiple tokens):

```
In·a·deep·bowl,·mix·the·orange·juice·with·the·sugar,·g
inger,·and·nutmeg.
```

Spanish: 33 tokens; 6/16 words are split

```
En·un·recipiente·hondo,·mezclar·el·jugo·de·naranja·con
·el·azúcar,·jengibre,·y·nuez·moscada.
```

Words and Tokens

Byte Pair Encoding

Words and Tokens

Corpora

Corpora

Words don't appear out of nowhere!

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.

Corpora vary along dimensions like

Language: 7097 languages in the world

It's important to test algorithms on multiple languages

What may work for one may not work for another

Corpora vary along dimensions like

Variety, like African American English varieties

- AAE Twitter posts might include forms like "*iont" (I don't)*

Genre: newswire, fiction, scientific articles, Wikipedia

Author Demographics: writer's age, gender, ethnicity, socio-economic status

Code Switching

Speakers use multiple languages in the same utterance

This is very common around the world

Especially in spoken language and related genres like texting and social media

Code Switching: Spanish/English

Por primera vez veo a @username actually being hateful! It was beautiful:)

[For the first time I get to see @username actually being hateful! it was beautiful:)]

Code Switching: Hindi/English

dost tha or ra- hega ... dont wory ... but dherya rakhe

[“he was and will remain a friend ... don’t worry ... but have faith”]

Corpus **datasheets**

Gebru et al (2020), Bender and Friedman (2018)

Motivation:

- Why was the corpus collected?
- By whom?
- Who funded it?

Situation: In what situation was the text written?

Collection process: How was it sampled? Was there consent? Pre-processing?

+Annotation process, variety, demographics, etc.

Words and Tokens

Corpora

Words and Tokens

Regular Expressions

Regular expressions are used everywhere

- A formal language for specifying text strings
- Part of every text processing task
  - Useful pre-processing or text formatting step, for example for BPE pre-tokenization
- Also necessary for data analysis of text
- A widely used tool in industry and academics

Regular expressions

We can use regular expressions to search for a pattern in a string

For example, the **Python** function

```
re.search(pattern,string)
```

scans through the string and returns the first match inside it for the pattern

Python syntax

We'll show regex on raw strings with double quotes:

```
r"regex"
```

Raw strings treat backslashes as literal characters

Many regex patterns use backslashes.

Why raw strings?

- Regex and Python both use backslash "\" for special characters. If you don't use raw notation you would have to type extra backslashes!
  - `"\\d+"` to search for 1 or more digits
  - `"\n"` in Python means the "newline" character, not a "slash" followed by an "n". Need `"\\n"` for two characters.
- Instead: use Python's **raw string notation** for regex:
  - `r"[tT]he"`
  - `r"\d+"` matches one or more digits
    - instead of `\\d+`

Regular expressions

The pattern

```
r"Buttercup"
```

matches the substring Buttercup in any string, like the string

```
I'm called little Buttercup
```

Regular Expressions: Disjunctions

Letters inside square brackets []

Pattern	Matches
`r"[mM]ary"`	`Mary` or `mary`
`r"[1234567890]"`	Any one digit

Ranges using the dash `[A-Z]`

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

Regular Expressions: Negation in Disjunction

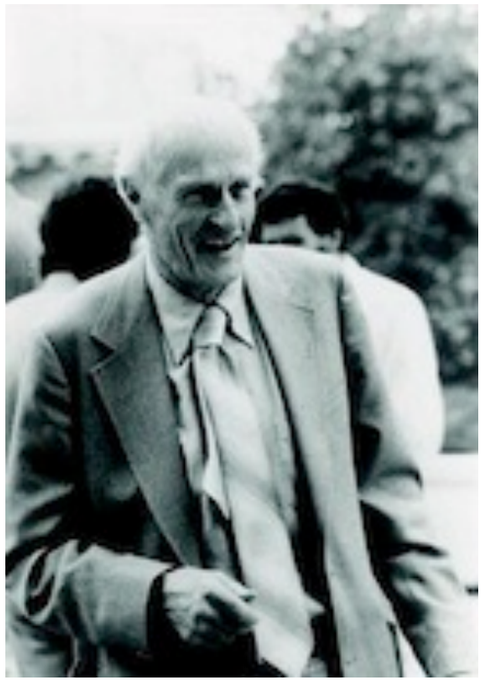
Carat as first character in [] negates the list

- Note: Carat means negation only when it's first in []
- Special characters (., *, +, ?) lose their special meaning inside []

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

Kleene star and Kleene plus

```
baa
baaa
baaaa  ...
```



Stephen C Kleene

Kleene star * (0 or more of previous characters)

Kleene plus + (1 or more of previous character)

```
r"baaa*"      baa baaa baaaa ...
r"baa+"
```

Wildcard

The period means "any character"

`r"."` matches anything

`r".*"` matches any sequence of 0 or more of anything

Regular Expressions: Anchors ^ $

Pattern	Matches
r"^[A-Z]"	Palo Alto
r"\.$"	The end.
r".$"	The end? The end!

Regular Expressions: More Disjunction

Groundhog is another name for woodchuck!

The pipe symbol | for disjunction

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



Regular Expressions: Convenient aliases

Pattern	Expansion	Matches	Examples
`r"\d"`	`[0-9]`	Any digit	`Fahreneit 451`
`r"\D"`	`[^0-9]`	Any non-digit	`Blue Moon`
`r"\w"`	`[a-ZA-Z0-9_]`	Any alphanumeric or _	`Daiyu`
`r"\W"`	`[^\w]`	Not alphanumeric or _	`Look!`
`r"\s"`	`[ \r\t\n\f]`	Whitespace (space, tab)	`Look␣up`
`r"\S"`	`[^\s]`	Not whitespace	`Look up`

The iterative process of writing regex's

Find me all instances of the word “the” in a text.

`the`

Misses capitalized examples

`[tT]he`

Incorrectly returns `other` or `Theology`

`\W[tT]he\W`

False positives and false negatives

The process we just went through was based on fixing two kinds of errors:

1. Not matching things that we should have matched (The)

   False negatives

2. Matching strings that we should not have matched (there, then, other)

   False positives

Characterizing work on NLP

In NLP we are always dealing with these kinds of errors.

Reducing the error rate for an application often involves two antagonistic efforts:

- Increasing coverage (or *recall*) (minimizing false negatives).
- Increasing accuracy (or *precision*) (minimizing false positives)

Regular expressions play a surprisingly large role

Widely used in both academics and industry

1. Part of most text processing tasks, even for big neural language model pipelines
   - including text formatting and pre-processing
2. Very useful for data analysis of any text data

Words and Tokens

Regular Expressions

Words and Tokens

Substitutions, Capture Groups, and Lookahead

Regex Substitutions in Python

To change every instance of cherry to apricot in string:

```
re.sub(r"cherry", r"apricot", string)
```

Upper case all examples of a name:

```
re.sub(r"janet", r"Janet", string)
```

Substitutions often need capture groups

Task: Change:

US format dates (mm/dd/yyyy) to

EU format dates (dd-mm-yyyy)

Pattern to match US format:

```
r"\d{2}/\d{2}/\d{4}"
```

How to specify in the *replacement* that we want to swap the date and month values?

Capture group

Use parentheses to capture (*store*) the values that we matched in the search,

Groups have numbers

In the replacement ("repl"), we refer back to that group with a number command.

Capture group

```
re.sub(r"(\d{2})/(\d{2})/(\d{4})",
r"\2-\1-\3", string)}
```

Parens (and) around the two month digits, the two day digits, and the four year digits,

This stores

- the first 2 digits in group 1,
- the second 2 digits in group 2,
- final digits in group 3.

Then in the *repl* string,

- \1, \2, and \3, refer to the 1st, 2nd, and 3rd registers.

That regex will

map

The date is 10/15/2011

to

The date is 15-10-2011

But suppose we don't want to capture?

Parentheses have a double function: grouping terms, and capturing

Non-capturing groups: add a ?: after paren:

```
r"(?:some|a few) (people|cats) like some \1/"
```

matches

- some cats like some cats

but not

- some cats like some some

Non-capture groups are usually used when we are trying to capture only part of a long or complex pattern.
Perhaps we are matching a sequence of dates (\d\d/\d\d/\d\d\d\d) separated by spaces and we want to extract only the 15th one.
We need to use parenthesis in order to use the counting
operator on the first 14, but we don’t want to store all the
useless information.
The following pattern only stores the 15th date in group 1:

r"(?:\d\d/\d\d/\d\d\d\d\s+){14}(\d\d/\d\d/\d\d\d\d)"

Lookahead assertions

`(?= pattern)` is true if pattern matches, but is **zero-width; doesn't advance character pointer**

`(?! pattern)` true if a pattern does not match

How to capture the first word on the line, but only if it doesn't start with the letter T:

```
r"^(?![tT])(\w+)\b"
```

Simple Application: ELIZA

Early NLP system that imitated a Rogerian psychotherapist

- Joseph Weizenbaum, 1966.

Uses pattern matching to match, e.g.,:

- `"I need X"`

and translates them into, e.g.

- `"What would it mean to you if you got X?`

Simple Application: ELIZA

Men are all alike.
IN WHAT WAY

They're always bugging us about something or other. CAN YOU THINK OF A SPECIFIC EXAMPLE

Well, my boyfriend made me come here.
YOUR BOYFRIEND MADE YOU COME HERE

He says I'm depressed much of the time.
I AM SORRY TO HEAR YOU ARE DEPRESSED

How ELIZA works

```
 re.sub(r".* I'M (depressed|sad) .*",r"I AM SORRY TO HEAR YOU ARE \1",input)
 re.sub(r".* I AM (depressed|sad) .*",r"WHY DO YOU THINK YOU ARE \1",input)


 re.sub(r".* all .*",r"IN WHAT WAY?",input)
 re.sub(r".* always .*",r"CAN YOU THINK OF A SPECIFIC EXAMPLE?",input)
```

Words and Tokens

Substitutions, Capture Groups, and Lookahead

Words and Tokens

Pretokenization for BPE

Pretokenization for BPE

The GPE-2 rules for pretokenization:

- Count clitics like 't and 'm as tokens
- Pull off punctuation and numbers

We do this with one regex:

```
r"'s|'t|'re|'ve|'m|'ll|'d| ?\p{L}+| ?\p{N}+| ?[^\s\p{L}\p{N}]+|\s+(?
```

The Python regex (as opposed to re) library

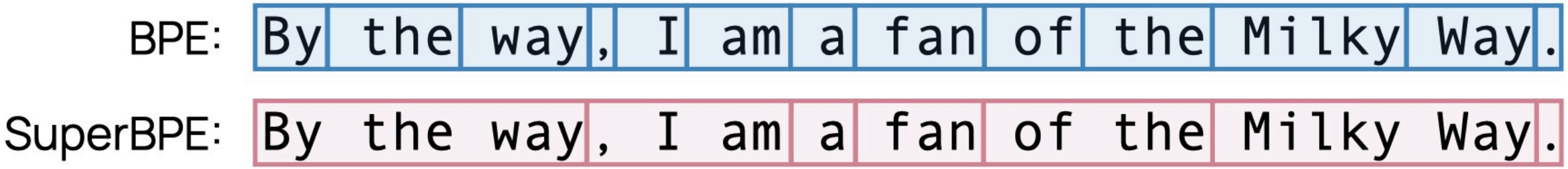
Special \p and \P operators:

\p{L} matches any Unicode letter,
\P{L} matches any non-letter,
\p{N} matches any number,
\P{N} matches any non-number.

```
>>> import regex as re
>>> pat = re.compile(
... # Contractions: 't and 'm are tokens
...     r"'s|'t|'re|'ve|'m|'ll|'d|"
... # Words:  sequence of Unicode letters (after optional space)
...     r" ?\p{L}+|"
... #Number: sequence of digits (after optional space)
...     r" ?\p{N}+|"
... # Punctuation: sequence of non-alphanumeric/non-space
...                          #(after optional space)
...     r" ?[^\s\p{L}\p{N}]+|"
... # whitespace
...     r"\s+(?!\S)|\s+"
... )
>>> text = "We're 350 dogs! Um, lunch?"
>>> print(pat.findall(text))
['We', "'re", ' 350', ' dogs', '!', ' Um', ',', ' lunch', '?']
>>>
```

Pretokenization could also allow multiple words!

SUPERBPE runs a second stage of BPE allowing merges across spaces and punctuation.



Liu et al 2025

Words and Tokens

Pretokenization for BPE

Words and Tokens

Rule-based tokenization and Simple Unix tools

Rule-based tokenization

Although subword tokenization is the norm

Sometimes we need particular tokens

Like for parsing, where the parser needs grammatical words, or social science

Issues for rule-based tokenization

Mostly but not always remove punctuation:

- m.p.h., Ph.D., AT&T, cap'n
- prices ($45.55)
- dates (01/02/06)
- URLs (http://www.stanford.edu)
- hashtags (#nlproc)
- email addresses (someone@cs.colorado.edu)

Numbers are tokenized differently across languages

- English 555,500.50 = French 555 500,50

Multiword expressions (MWE)?

- New York, rock 'n' roll

Penn Treebank Tokenization Standard

Input: "The San Francisco-based restaurant," they said,
"doesn't charge $10".

Output: "␣The␣San␣Francisco-based␣restaurant␣,␣"␣they␣said␣,␣
"␣does␣n't␣charge␣$␣10␣"␣.

Tokenization in NLTK

Bird, Loper and Klein (2009), *Natural Language Processing with Python*. O'Reilly

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)    # set flag to allow verbose regexps
...     (?:[A-Z]\.)+        # abbreviations, e.g. U.S.A.
...   | \w+(?:-\w+)*        # words with optional internal hyphens
...   | \$?\d+(?:\.\d+)?%?  # currency, percentages, e.g. $12.40, 82%
...   | \.\.\.              # ellipsis
...   | [][.,;"'?():_`-]  # these are separate tokens; includes ], [
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

Sentence Segmentation

!, ? mostly unambiguous but **period** "." is very ambiguous

- Sentence boundary
- Abbreviations like Inc. or Dr.
- Numbers like .02% or 4.3

Common algorithm: Tokenize first: use rules or ML to classify a period as either (a) part of the word or (b) a sentence-boundary.

- An abbreviation dictionary can help

Sentence segmentation can then often be done by rules based on this tokenization.

Space-based tokenization

A very simple way to tokenize

- For languages that use space characters between words
  - Arabic, Cyrillic, Greek, Latin, etc., based writing systems
- Segment off a token between instances of spaces

Unix tools for space-based tokenization

- The "tr" command
- Inspired by Ken Church's UNIX for Poets
- Given a text file, output the word tokens and their frequencies

Simple Tokenization in UNIX

(Inspired by Ken Church's UNIX for Poets.)

Given a text file, output the word tokens and their frequencies

```
tr -sc 'A-Za-z' '\n' < shakes.txt
      | sort
      | uniq -c
```

Change all non-alpha to newlines

Sort in alphabetical order

Merge and count each type

```
1945 A
  72 AARON
  19 ABBESS
   5 ABBOT
 ... ...
```

```
25 Aaron
 6 Abate
 1 Abates
 5 Abbess
 6 Abbey
 3 Abbot
... ...
```

The first step: tokenizing

```
tr -sc 'A-Za-z' '\n' < shakes.txt | head
```

```
THE
SONNETS
by
William
Shakespeare
From
fairest
creatures
We
...
```

The second step: sorting

```
tr -sc 'A-Za-z' '\n' < shakes.txt | sort | head
```

```
A
A
A
A
A
A
A
A
A
...
```

More counting

Merging upper and lower case

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c
```

Sorting the counts

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c | sort -n -r
```

```
  23243 the
  22225 i
  18618 and
  16339 to
  15687 of
  12780 a
  12163 you
  10839 my
  10005 in
   8954  d
```

What happened here?

Words and Tokens

Rule-based tokenization and Simple Unix tools



Minimum Edit Distance

Definition of Minimum Edit Distance



How similar are two strings?

- Spell correction
  - The user typed “graffe”

  Which is closest?
    - graf
    - graft
    - grail
    - giraffe

- Computational Biology
  - Align two sequences of nucleotides

    AGGCTATCACCTGACCTCCAGGCCGATGCCC
    TAGCTATCACGACCGCGGTCGATTTGCCCGAC

  - Resulting alignment:

    –AGG**CTATCAC**C**TGACCT**C**A**G**G**C**CGA--TGCCC**---
    TAG–CTATCAC--GACCG**C--GG**T**CGA**TT**TGCCC**GAC

- Also for Machine Translation, Information Extraction, Speech Recognition



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



Minimum Edit Distance

- Two strings and their **alignment**:

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



Minimum Edit Distance

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

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



Alignment in Computational Biology

- Given a sequence of bases

AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGTCGATTTGCCCGAC

- An alignment:

–AGG**CTATCAC**C**T****GACC**T**C**C**A****GG**C**CGA****––TGCCC****–––**
T**AG****–****CTATCAC****––****GACC**G**C****––****GG**T**CGA**T**T****TGCCC**GAC

- Given two sequences, align each letter to a letter or gap



Other uses of Edit Distance in NLP

- Evaluating Machine Translation and speech recognition

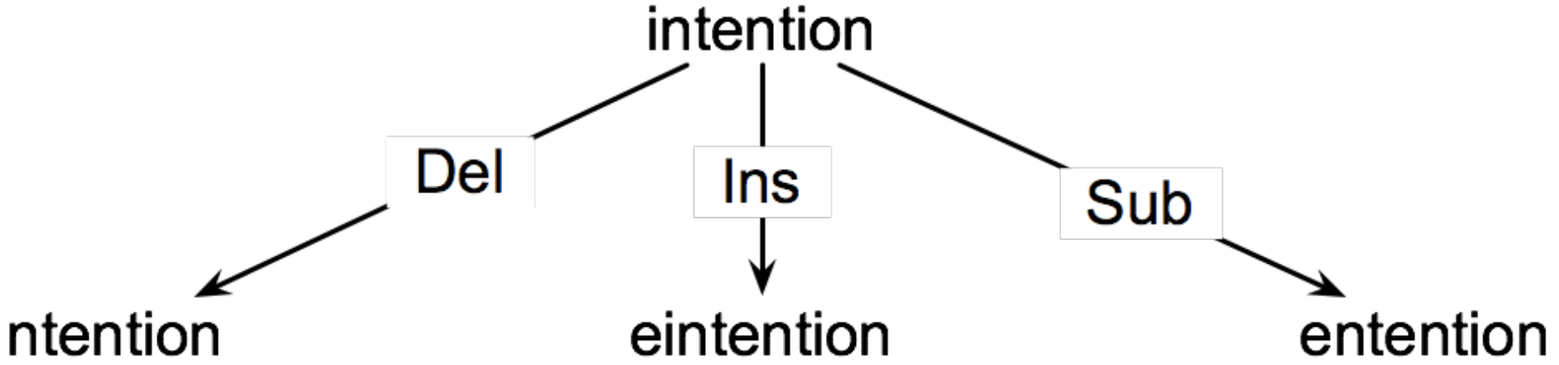
```
R Spokesman confirms    senior government adviser was shot
H Spokesman said    the senior          adviser was shot dead
              S     I             D                     I
```

- Named Entity Extraction and Entity Coreference
  - IBM Inc. announced today
  - IBM profits
  - Stanford President John Hennessy announced yesterday
  - for Stanford University President John Hennessy



How to find the Min Edit Distance?

- Searching for a path (sequence of edits) from the start string to the final string:
  - **Initial state**: the word we're transforming
  - **Operators**: insert, delete, substitute
  - **Goal state**: the word we're trying to get to
  - **Path cost**: what we want to minimize: the number of edits





Minimum Edit as Search

- But the space of all edit sequences is huge!
  - We can't afford to navigate naïvely
  - Lots of distinct paths wind up at the same state.
    - We don't have to keep track of all of them
    - Just the shortest path to each of those revisted states.



Defining Min Edit Distance

- For two strings
  - X of length *n*
  - Y of length *m*
- We define D(*i,j*)
  - the edit distance between X[1..*i*] and Y[1..*j*]
    - i.e., the first *i* characters of X and the first *j* characters of Y
  - The edit distance between X and Y is thus D(*n,m*)



Minimum Edit Distance

Definition of Minimum Edit Distance



Minimum Edit Distance

Computing Minimum Edit Distance



Dynamic Programming for Minimum Edit Distance

- **Dynamic programming**: A tabular computation of D(*n*,*m*)
- Solving problems by combining solutions to subproblems.
- Bottom-up
  - We compute D(i,j) for small *i*,*j*
  - And compute larger D(i,j) based on previously computed smaller values
  - i.e., compute D(*i*,*j*) for all *i* (0 < *i* < n) and *j* (0 < j < m)



Defining Min Edit Distance (Levenshtein)

- Initialization

  $D(i,0) = i$

  $D(0,j) = j$

- Recurrence Relation:

  For each $i = 1 \ldots M$

  For each $j = 1 \ldots N$

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

- Termination:

  $D(N,M)$ is distance



The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N



The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$



Edit Distance

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N



The Edit Distance Table

N	9	8	9	10	11	12	11	10	9	**8**
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N



Minimum Edit Distance

Computing Minimum Edit Distance



Minimum Edit Distance

Backtrace for Computing Alignments



Computing alignments

- Edit distance isn't sufficient
  - We often need to **align** each character of the two strings to each other
- We do this by keeping a "backtrace"
- Every time we enter a cell, remember where we came from
- When we reach the end,
  - Trace back the path from the upper right corner to read off the alignment



Edit Distance

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N



MinEdit with Backtrace

n	9	↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↙←↓ 12	↓ 11	↓ 10	↓ 9	↙ **8**
o	8	↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↓ 10	↓ 9	↙ **8**	← 9
i	7	↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↓ 9	↙ **8**	← 9	← 10
t	6	↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↙ **8**	← 9	← 10	←↓ 11
n	5	↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ **8**	↙←↓ 9	↙←↓ 10	↙←↓ 11	↙↓ 10
e	4	↙ 3	← 4	↙← **5**	← **6**	← 7	←↓ 8	↙←↓ 9	↙←↓ 10	↓ 9
t	3	↙←↓ 4	↙←↓ **5**	↙←↓ 6	↙←↓ 7	↙←↓ 8	↙ 7	←↓ 8	↙←↓ 9	↓ 8
n	2	↙←↓ **3**	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↓ 7	↙←↓ 8	↙ 7
i	**1**	↙←↓ 2	↙←↓ 3	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙ 6	← 7	← 8
#	**0**	1	2	3	4	5	6	7	8	9
	#	**e**	**x**	**e**	**c**	**u**	**t**	**i**	**o**	**n**



Adding Backtrace to Minimum Edit Distance

- Base conditions:

  $D(i,0) = i$ $\qquad$ $D(0,j) = j$

  Termination:

  $D(N,M)$ is distance

- Recurrence Relation:

For each $i = 1 \ldots M$

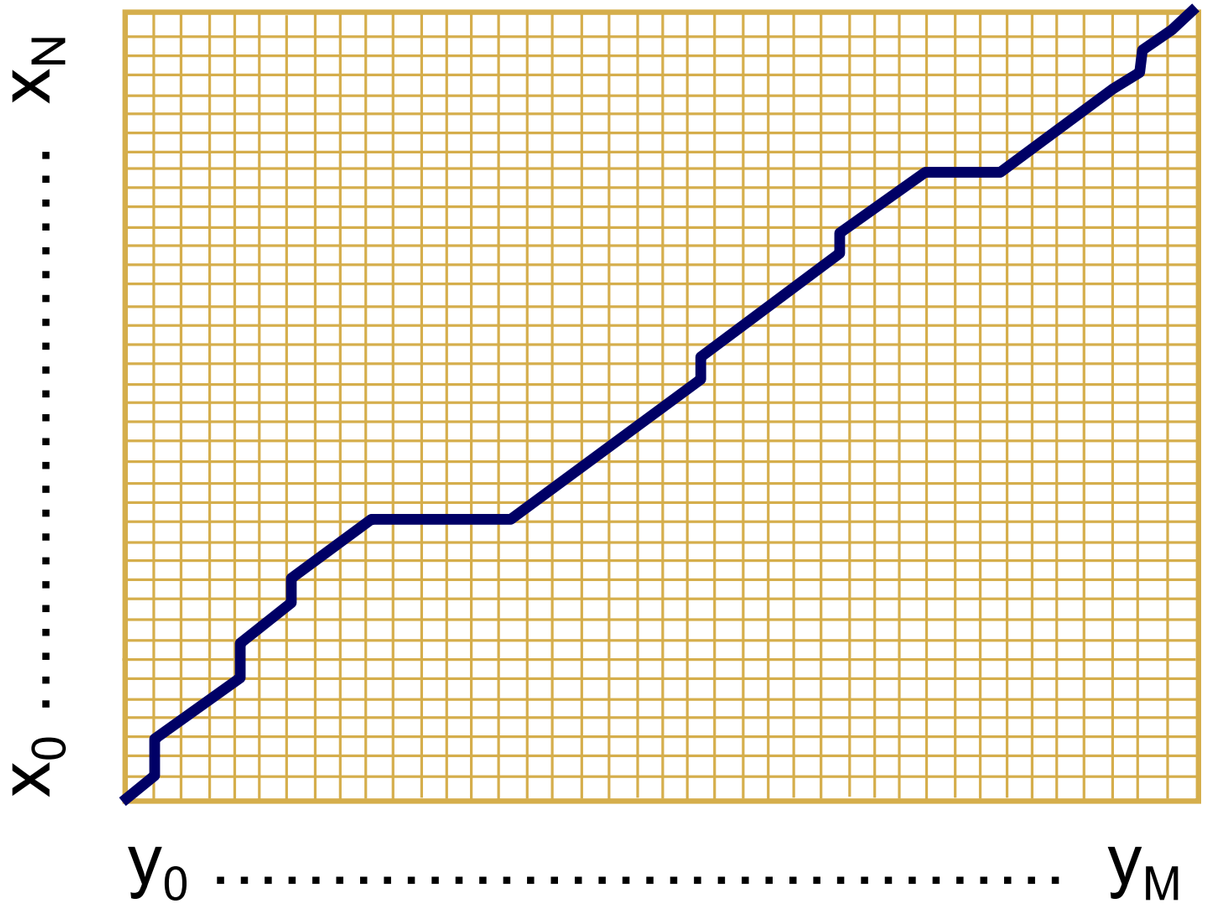
For each $j = 1 \ldots N$

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 & \text{deletion} \\ D(i,j-1) + 1 & \text{insertion} \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \quad \text{substitution} \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

$$ptr(i,j) = \begin{cases} \text{LEFT} & \text{insertion} \\ \text{DOWN} & \text{deletion} \\ \text{DIAG} & \text{substitution} \end{cases}$$



The Distance Matrix



Every non-decreasing path

from (0,0) to (M, N)

corresponds to
an alignment
of the two sequences

An optimal alignment is composed of optimal subalignments

Slide adapted from Serafim Batzoglou



Result of Backtrace

- Two strings and their **alignment**:

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



Performance

- Time:

  O(nm)

- Space:

  O(nm)

- Backtrace

  O(n+m)



Minimum Edit Distance

Backtrace for Computing Alignments



Minimum Edit Distance

Weighted Minimum Edit Distance



Weighted Edit Distance

- Why would we add weights to the computation?
  - Spell Correction: some letters are more likely to be mistyped than others
  - Biology: certain kinds of deletions or insertions are more likely than others



Confusion matrix for spelling errors

sub[X, Y] = Substitution of X (incorrect) for Y (correct)

X	Y (correct)																								
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	7	1	342	0	0	2	118	0	1	0	0	3	76	0	0	1	35	9	9	0	1	0	5	0
b	0	0	9	9	2	2	3	1	0	0	0	5	11	5	0	10	0	0	2	1	0	0	8	0	0	0
c	6	5	0	16	0	9	5	0	0	0	1	0	7	9	1	10	2	5	39	40	1	3	7	1	1	0
d	1	10	13	0	12	0	5	5	0	0	2	3	7	3	0	1	0	43	30	22	0	0	4	0	2	0
e	388	0	3	11	0	2	2	0	89	0	0	3	0	5	93	0	0	14	12	6	15	0	1	0	18	0
f	0	15	0	3	1	0	5	2	0	0	0	3	4	1	0	0	0	6	4	12	0	0	2	0	0	0
g	4	1	11	11	9	2	0	0	0	1	1	3	0	0	2	1	3	5	13	21	0	0	1	0	3	0
h	1	8	0	3	0	0	0	0	0	0	2	0	12	14	2	3	0	3	1	11	0	0	2	0	0	0
i	103	0	0	0	146	0	1	0	0	0	0	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0
j	0	1	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	0	5	0	0	0	0	0	0	0
k	1	2	8	4	1	1	2	5	0	0	0	0	5	0	2	0	0	0	6	0	0	0	4	0	0	3
l	2	10	1	4	0	4	5	6	13	0	1	0	0	14	2	5	0	11	10	2	0	0	0	0	0	0
m	1	3	7	8	0	2	0	6	0	0	4	4	0	180	0	6	0	0	9	15	13	3	2	2	3	0
n	2	7	6	5	3	0	1	19	1	0	4	35	78	0	0	7	0	28	5	7	0	0	1	2	0	2
o	91	1	1	3	116	0	0	0	25	0	2	0	0	0	0	14	0	2	4	14	39	0	0	0	18	0
p	0	11	1	2	0	6	5	0	2	9	0	2	7	6	15	0	0	1	3	6	0	4	1	0	0	0
q	0	0	1	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	14	0	30	12	2	2	8	2	0	5	8	4	20	1	14	0	0	12	22	4	0	0	1	0	0
s	11	8	27	33	35	4	0	1	0	1	0	27	0	6	1	7	0	14	0	15	0	0	5	3	20	1
t	3	4	9	42	7	5	19	5	0	1	0	14	9	5	5	6	0	11	37	0	0	2	19	0	7	6
u	20	0	0	0	44	0	0	0	64	0	0	0	0	2	43	0	0	4	0	0	0	0	2	0	8	0
v	0	0	7	0	0	3	0	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	0	7	0	6	3	3	1	0	0	0	0	0
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0
y	0	0	2	0	15	0	1	7	15	0	0	0	2	0	6	1	0	7	36	8	5	0	0	1	0	0
z	0	0	0	7	0	0	0	0	0	0	0	7	5	0	0	0	0	2	21	3	0	0	0	0	3	0


~ `
! 1
@ 2
# 3
$ 4
% 5
^ 6
& 7
* 8
(9
) 0
— -
+ =
Tab
Q W E R T Y U I O P
{ [
}]
Caps Lock
A S D F G H J K L
: ;
" '
Enter
Shift
Z X C V B N M
< ,
> .
? /
Shift
Ctrl
Alt
Alt



Weighted Min Edit Distance

- Initialization:

$$D(0,0) = 0$$
$$D(i,0) = D(i-1,0) + del[x(i)]; \quad 1 < i \leq N$$
$$D(0,j) = D(0,j-1) + ins[y(j)]; \quad 1 < j \leq M$$

- Recurrence Relation:

$$D(i,j) = \min \begin{cases} D(i-1,j) + del[x(i)] \\ D(i,j-1) + ins[y(j)] \\ D(i-1,j-1) + sub[x(i),y(j)] \end{cases}$$

- Termination:

$$D(N,M) \text{ is distance}$$



Where did the name, dynamic programming, come from?

…The 1950s were not good years for mathematical research. [the] Secretary of Defense …had a pathological fear and hatred of the word, research…

I decided therefore to use the word, "**programming**".

I wanted to get across the idea that this was dynamic, this was multistage… I thought, let's … take a word that has an absolutely precise meaning, namely **dynamic**… it's impossible to use the word, **dynamic**, in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible.

Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to."

Richard Bellman, "Eye of the Hurricane: an autobiography" 1984.


sentences
shows
probability
grammar
training
model
english
first
state
information
speech
set
algorithm
words
feature
verb
representation
structure
semantic
different
features
rules
languages
corpus
language
system
hmm
sentence
equation
number
processing
given
using
dialogue
models
observation
relation
noun
parsing
based
just
three
text
recall
rule
systems
machine
meaning
search
simple
vector
expression
similar
input
recognition
parse
since
problem
instead
learning
start
either
tagging
human
unification
natural
form
process
represent
analysis
tag
regular
test
translation
relations
np
lexical
complex
introduced
expressions
part-of-speech
similarity
type
terms
relation
sense
discourse
probabilistic
case
constraints
tree
note
ledge
sequence
output
value
compute
maximum
general
subject
earlier
decoding
syntactic
morphological
occur
phrases
viterbi
distance
want
representations
role
class
finite-state
semantics
means
acoustic
np
models
observation
noun
sentence
equation
processing
using
dialogue
given
number
regular
corpus
consider
language
rules
form
hmm
tag
system
structure
features
semantic
different
languages
unification
representation
human
tagging
verb
feature
algorithm
input
recognition
parse
since
problem
instead
meaning
search
simple
vector
expression
similar
role
class
learning
start
either
machine
rule
morphological
syntactic
systems
speech
set
state
information
sequence
compute
value
output
maximum
general
subject
earlier
decoding
text
recall
three
first
training
phrase
likelihood
find
user
discussed
word
next
rather
verbs
grammar
probability
sentences
shows
context
data
frequency
event
whether
classifier
flight
final
method
important
question
probabilistic
sense
discourse
case
parsing
based
just
three
text
note
ledge
tree
constraints
sequence
state
information
speech
set
algorithm
words
feature
verb
representation
structure
semantic
different
features
rules
languages
corpus
language
system
hmm

ACKNOWLEDGEMENT & SOURCE ATTRIBUTION

- The teaching materials presented in these slides have been developed with reference to the textbook and lecture presentation materials of:
- **Daniel Jurafsky and James H. Martin**
- *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models.*
- **Third Edition – Online Draft, Stanford University.**
- Official resource: https://web.stanford.edu/~jurafsky/slp3/
- We gratefully acknowledge the authors for making their textbook, lecture slides, and educational resources available for academic and teaching purposes.
- **Note:** Certain concepts, explanations, examples, illustrations, and figures have been adapted or reproduced from the above reference. Additional explanations, examples, and teaching materials have been incorporated for classroom instruction.
- **Prepared for academic and educational purposes.**

Module 1 Questionnaires

1. Define Natural Language Processing (NLP). Explain its objectives and the importance of developing computational models for human language processing. *[6 Marks | L2 | CO1]*
2. Differentiate between Rationalist and Empiricist approaches to NLP. Discuss how each approach contributes to understanding human language acquisition. *[8 Marks | L2 | CO1]*
3. Define Computational Linguistics. Explain its relationship with theoretical linguistics and psycholinguistics with suitable examples. *[6 Marks | L2 | CO1]*
4. Explain knowledge-driven and data-driven computational models of NLP. Compare their characteristics, advantages and limitations. *[8 Marks | L2 | CO1]*
5. Explain lexical analysis and the role of morphological knowledge in identifying the structure and formation of words. Illustrate with suitable examples. *[6 Marks | L2 | CO1]*
6. Explain syntactic analysis in NLP. Illustrate how syntactic rules help identify the grammatical structure of a sentence using suitable examples. *[6 Marks | L2 | CO1]*
7. Explain semantic analysis and its importance in identifying the meaning of words and sentences. Illustrate with a suitable example of semantic anomaly. *[6 Marks | L2 | CO1]*
8. Explain pragmatic analysis in NLP. How does contextual knowledge help in identifying the intended meaning of an utterance? Illustrate with an example. *[6 Marks | L2 | CO1]*
9. Explain discourse analysis and the role of anaphoric reference resolution in understanding paragraphs and documents with suitable examples. *[6 Marks | L2 | CO1]*
10. Describe the different levels of language processing: lexical, syntactic, semantic, pragmatic and discourse analysis. Explain the type of linguistic knowledge required at each level with suitable examples. *[8 Marks | L2 | CO1]*
11. Consider the sentence "Can you open the window?" Explain how semantic and pragmatic analysis interpret this sentence differently. Discuss the importance of context in determining the speaker's intention. *[6 Marks | L2 | CO1]*
12. Consider the sentence "The district administration refused to give the trade union permission for the meeting because they feared violence." Explain the role of discourse and pragmatic analysis in resolving the anaphoric reference "they". *[6 Marks | L2 | CO1]*
13. Explain the major challenges involved in Natural Language Processing. Discuss ambiguity, vagueness, incomplete knowledge and the evolving nature of language. *[8 Marks | L2 | CO1]*
14. What is lexical ambiguity? Explain how a word can have multiple meanings depending on its context with suitable examples. *[6 Marks | L2 | CO1]*
15. Differentiate between lexical ambiguity and structural ambiguity in NLP with suitable examples. *[6 Marks | L2 | CO1]*
16. Explain part-of-speech ambiguity and word sense ambiguity. Illustrate how the words "can" and "bank" can have different interpretations depending on their usage. *[6 Marks | L2 | CO1]*
17. Explain structural ambiguity in natural language. Illustrate the two possible interpretations of the sentence "I saw the man with a telescope." *[6 Marks | L2 | CO1]*
18. Explain how idioms and metaphors create challenges for NLP systems in understanding the intended meaning of natural language expressions. Give suitable examples. *[6 Marks | L2 | CO1]*
19. What is ellipsis in natural language? Explain how omission of words or phrases creates difficulties in NLP with suitable examples. *[6 Marks | L2 | CO1]*
20. Explain how context and co-occurring words help NLP systems identify the intended meaning of ambiguous words. Illustrate with suitable examples. *[6 Marks | L2 | CO1]*
21. Explain why grammar alone is insufficient to understand natural language. Illustrate your answer using the sentences "Colorless green ideas sleep furiously" and "Me go home now." *[6 Marks | L2 | CO1]*
22. Explain the limitations of grammar-based NLP systems in handling flexible sentence structures, informal language and context-dependent meanings. *[6 Marks | L2 | CO1]*
23. Define language grammar. Explain the importance of grammatical rules in parsing and generating natural language sentences. *[6 Marks | L2 | CO1]*
24. Explain Context-Free Grammar (CFG), Dependency Grammar and Phrase Structure Grammar with suitable examples. *[8 Marks | L2 | CO1]*

25. Differentiate between Context-Free Grammar, Dependency Grammar and Phrase Structure Grammar based on their representation of sentence structure. *[6 Marks | L2 | CO1]*
26. Explain the contribution of Noam Chomsky to generative grammar and its significance in Natural Language Processing. *[6 Marks | L2 | CO1]*
27. Explain Chomsky's concept of deep structure and surface structure in transformational grammar with a suitable example. *[6 Marks | L2 | CO1]*
28. Explain the three components of transformational grammar: phrase structure grammar, transformational rules and morphophonemic rules. *[8 Marks | L2 | CO1]*
29. Explain how transformational rules convert an active sentence into a passive sentence using the example "The police will catch the snatcher." *[8 Marks | L2 | CO1]*
30. Explain the role of morphophonemic rules in transformational grammar. Illustrate how the transformation of "catch + en" results in "caught". *[6 Marks | L2 | CO1]*
31. Differentiate between deep structure and surface structure. Explain how sentences having different surface structures can convey the same meaning. *[6 Marks | L2 | CO1]*
32. Describe any four major applications of NLP with suitable real-world examples. *[8 Marks | L2 | CO1]*
33. Differentiate between words and tokens in NLP. Explain why identifying word boundaries is challenging across different languages. *[6 Marks | L2 | CO1]*
34. Explain the difference between vocabulary types and word instances in a text corpus with a suitable example. *[6 Marks | L2 | CO1]*
35. Explain the difference between function words and content words with suitable examples. Discuss their significance in NLP. *[6 Marks | L2 | CO1]*
36. Explain why conventional word-based tokenization is challenging for languages such as Chinese, Japanese and Thai. *[6 Marks | L2 | CO1]*
37. Explain Heaps' Law (Herdan's Law) and discuss how vocabulary size increases with corpus size. *[6 Marks | L2 | CO1]*
38. Explain the need for subword tokenization in NLP. Discuss the limitations of word-based and character-based tokenization approaches. *[8 Marks | L2 | CO1]*
39. Explain the problems associated with unknown words (OOV) in NLP and discuss how subword tokenization helps address these problems. *[6 Marks | L2 | CO1]*
40. Define morpheme and morphology. Explain how words are formed using morphemes with suitable examples. *[6 Marks | L2 | CO1]*
41. Differentiate between free and bound morphemes with suitable examples. *[6 Marks | L2 | CO1]*
42. Explain roots and affixes in morphological analysis. Illustrate how roots and affixes contribute to the meaning of a word. *[6 Marks | L2 | CO1]*
43. Differentiate between inflectional and derivational morphemes with suitable examples. *[6 Marks | L2 | CO1]*
44. Explain the concept of clitics in NLP. Illustrate how clitics differ from independent words and bound morphemes with suitable examples. *[6 Marks | L2 | CO1]*
45. Explain morphological typology and discuss the two important dimensions used to characterize languages: number of morphemes per word and ease of morpheme segmentation. *[6 Marks | L2 | CO1]*
46. Explain the classification of languages into analytic, synthetic and polysynthetic types based on the number of morphemes per word. *[6 Marks | L2 | CO1]*
47. Differentiate between agglutinative and fusion languages with suitable examples. Explain how their morphological structures affect tokenization. *[8 Marks | L2 | CO1]*
48. Consider the word "unbreakable". Perform morphological analysis by identifying its root and affixes. Explain the contribution of each morpheme to the overall meaning. *[6 Marks | L3 | CO1]*
49. What is Unicode? Explain its importance in representing text from different languages, scripts and symbol systems in NLP. *[6 Marks | L2 | CO1]*
50. Explain the limitations of ASCII encoding and justify the need for Unicode to represent multilingual text. *[6 Marks | L2 | CO1]*
51. Explain Unicode code points and their representation using hexadecimal notation. Illustrate with suitable examples. *[6 Marks | L2 | CO1]*
52. Differentiate between a Unicode code point and a glyph. Explain the role of fonts in displaying Unicode characters. *[6 Marks | L2 | CO1]*

53. Explain UTF-8 encoding and its variable-length encoding mechanism. Discuss how Unicode code points are represented using one, two, three or four bytes. *[8 Marks | L2 | CO1]*
54. Explain how UTF-8 maintains backward compatibility with ASCII and why it is suitable for multilingual text representation. *[6 Marks | L2 | CO1]*
55. Explain the advantages of UTF-8 encoding, including storage efficiency, self-synchronization and ASCII compatibility. *[6 Marks | L2 | CO1]*
56. Consider the English text "hello". Explain its Unicode code point representation and UTF-8 byte encoding. *[6 Marks | L2 | CO1]*
57. Explain why Unicode characters are encoded into multiple bytes in UTF-8 instead of storing every character using a fixed four-byte representation. *[6 Marks | L2 | CO1]*
58. Explain the need for subword tokenization in NLP. Discuss how it addresses the limitations of word-based, character-based and morpheme-based tokenization. *[8 Marks | L2 | CO1]*
59. Explain the two major components of subword tokenization: token learner and token encoder/segmenter. *[6 Marks | L2 | CO1]*
60. Explain the working principle of Byte-Pair Encoding (BPE) and how frequently occurring adjacent tokens are merged to construct a vocabulary. *[6 Marks | L2 | CO1]*
61. Explain the steps involved in the BPE token learner algorithm. Illustrate how the initial character vocabulary is expanded through repeated merging of frequent adjacent pairs. *[8 Marks | L2 | CO1]*
62. Consider the following corpus: "low low lower lowest". Apply the Byte-Pair Encoding algorithm by identifying frequent adjacent character pairs and illustrate the vocabulary formation through successive merges. *[8 Marks | L3 | CO1]*
63. Consider the following corpus: "banana banana band". Apply BPE for a specified number of merges. Show the pair frequencies, selected pairs and updated corpus after each merge. *[8 Marks | L3 | CO1]*
64. Given a corpus containing repeated character sequences, explain how BPE identifies the most frequent neighboring pair and replaces all occurrences with a newly merged token. Illustrate with a suitable example. *[6 Marks | L2 | CO1]*
65. Explain how BPE learns common subword units such as prefixes and suffixes from a training corpus. Illustrate how the prefix "re-" can be learned through repeated merging. *[6 Marks | L2 | CO1]*
66. Explain the BPE token encoder algorithm. How are the merge rules learned during training applied to tokenize an unseen test word? *[8 Marks | L2 | CO1]*
67. Differentiate between the BPE token learner and BPE token encoder with respect to their inputs, operations and outputs. *[6 Marks | L2 | CO1]*
68. Explain how BPE handles unseen words by decomposing them into known subword tokens. Illustrate with a suitable example. *[6 Marks | L2 | CO1]*
69. Explain rule-based tokenization in NLP. Discuss why specific tokenization rules are required for punctuation, abbreviations, numbers and special expressions. *[6 Marks | L2 | CO1]*
70. Explain the challenges involved in tokenizing abbreviations, currency values, dates, URLs, hashtags and email addresses with suitable examples. *[8 Marks | L2 | CO1]*
71. Define Regular Expressions. Explain their importance in NLP for text searching, preprocessing, formatting and information extraction. *[6 Marks | L2 | CO1]*
72. Explain the use of character classes and ranges in regular expressions with suitable examples such as [A-Z], [a-z] and [0-9]. *[6 Marks | L2 | CO1]*
73. Explain disjunction and negation in regular expressions using the symbols | and [^]. Illustrate with suitable examples. | 6 | L2 | CO1
74. Explain the working of Kleene star (*) and Kleene plus (+) operators in regular expressions with suitable examples. *[6 Marks | L2 | CO1]*
75. Explain the role of wildcard (.) and repetition (.*) operators in matching text patterns with suitable examples. *[6 Marks | L2 | CO1]*
76. Explain the use of anchors (^ and $) in regular expressions. Illustrate how they are used to match the beginning and end of a string. *[6 Marks | L2 | CO1]*
77. Explain the convenient aliases \d, \D, \w, \W, \s and \S used in regular expressions with suitable examples. *[8 Marks | L2 | CO1]*
78. Explain the working of Python’s re.search() function and the importance of raw strings (r"...") when writing regular-expression patterns. *[6 Marks | L2 | CO1]*

79. Write regular-expression patterns to extract email IDs, URLs and dates from a given text. Explain the pattern components used for each extraction. *[8 Marks | L3 | CO1]*
80. Explain how regular expressions can be designed to identify occurrences of the word "the" while reducing false positives and false negatives. *[8 Marks | L2 | CO1]*
81. Explain substitutions in regular expressions using Python's re.sub() function with suitable examples. *[6 Marks | L2 | CO1]*
82. Explain capture groups and non-capturing groups in regular expressions. Illustrate how captured groups can be used to rearrange date components. *[8 Marks | L2 | CO1]*
83. Explain positive and negative lookahead assertions in regular expressions with suitable examples. *[6 Marks | L2 | CO1]*
84. Explain how regular expressions are used in the ELIZA system for pattern matching and response generation. Illustrate with a suitable conversation example. *[8 Marks | L2 | CO1]*
85. Explain the role of regular expressions in BPE pre-tokenization. Discuss how words, numbers, punctuation and clitics are separated before applying BPE. *[8 Marks | L2 | CO1]*
86. Define Minimum Edit Distance. Explain how it measures the minimum number of editing operations required to transform one string into another. *[6 Marks | L2 | CO1]*
87. Explain insertion, deletion and substitution operations in Minimum Edit Distance with suitable examples. *[6 Marks | L2 | CO1]*
88. Explain the applications of Minimum Edit Distance in spell correction, machine translation, speech recognition*[6 Marks | L2 | CO1]*