6/08/22

Natural Language Processing

→ Branch of AI
→ Able to process human languages (English, Hindi, ...)
→ human - machine interaction

& Goals

1. Science goal
→ ~~Main~~ to understand how a given language operates.
→ Achieved using grammar

2. Engineering goal
→ to build systems that analyze and generate languages and reduces human-machine gap
→ Analyses patterns like
Subject verb object (structure of sentence)
(S) (V) (O).
→ Involves parsing
~~Involves~~ (Eg):
→ S + V + O
S → NP VP (NP - noun phrase
NP → N VP VP - verb phrase
VP → V NP

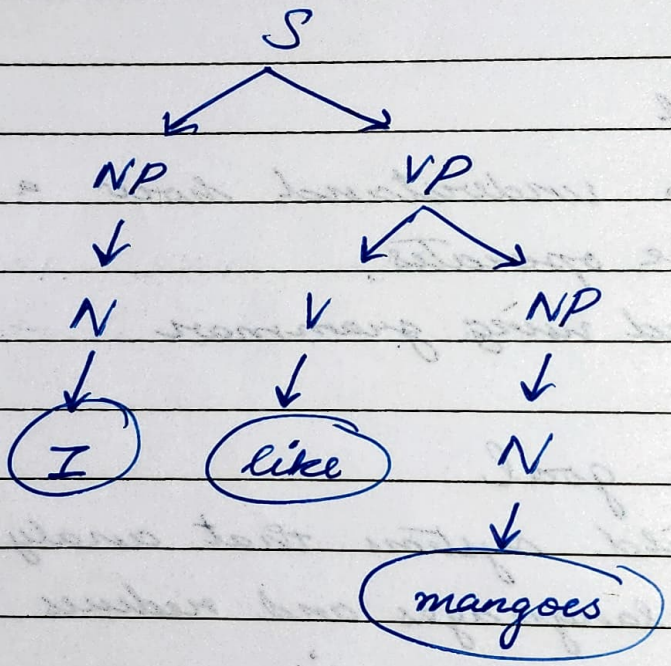
(Eg): I like mangoes

G1 : $S \rightarrow NP \quad VP$

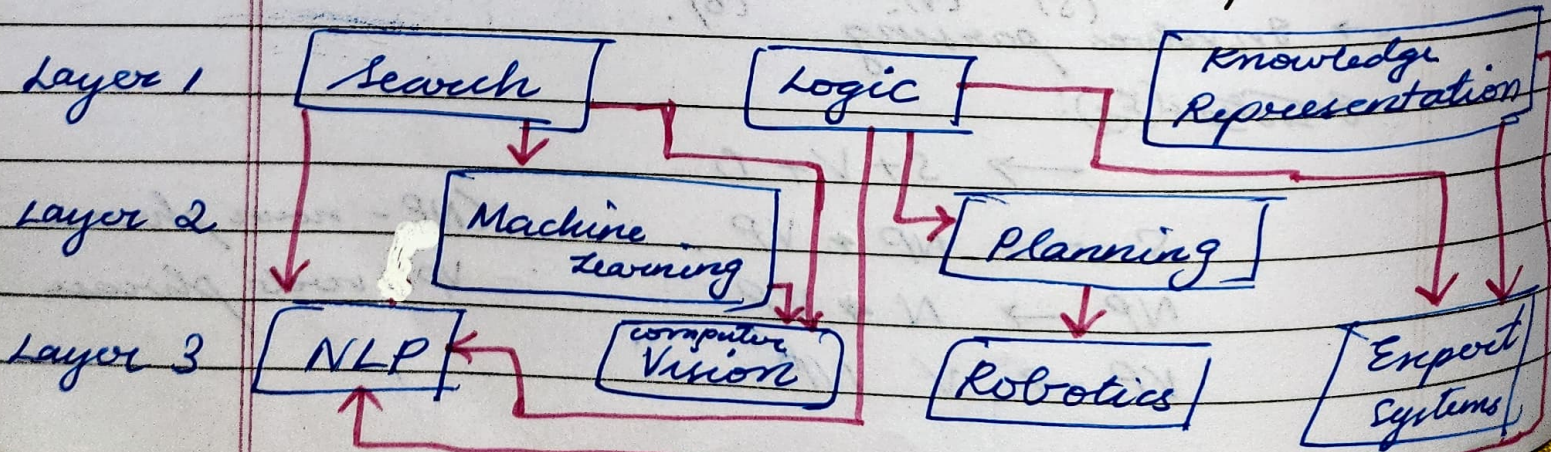
$NP \rightarrow N \quad VP$

$VP \rightarrow V \quad NP.$

Parse tree :



Areas of AI and their interdependencies



Stages of NLP

1. Phonology
2. Morphology
3. Lexical analysis
4. Syntax
5. Semantic
6. Pragmatics
7. Discourse

1) Phonology:

* Homophones:

Words that sound ~~similar~~ similar

(Eg): bank (can mean river's bank or financial bank).

* Near homophones:

Words that sound ~~similar~~ close to each other (near by)

(Eg): Kanna, Kannaa

*. Word Boundary: Breaking of sentences.

Syntax analysis / parsing

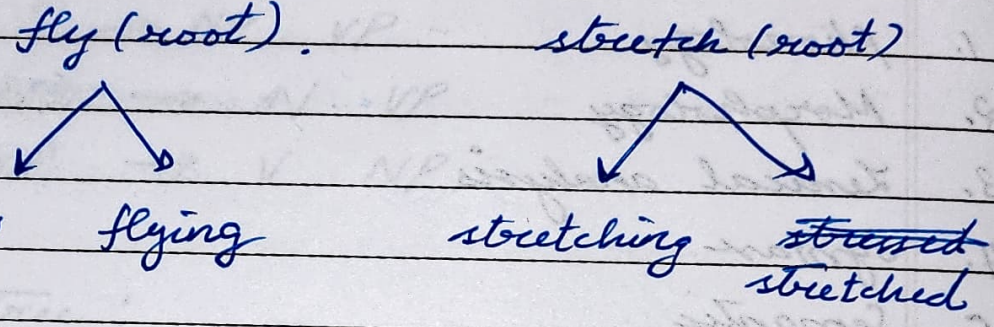
(Eg): I / like / mangoes
N / V / N

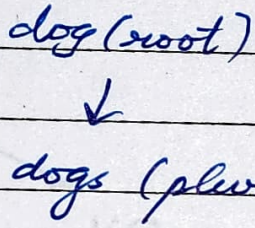
2). Morphology:

Formation of new words from root words

Implemented using finite automata (FSM)

(Eg):





3) Lexical Analysis :

→ Creating dictionaries

→ How the root words are stored in dictionaries.

(Eg): What is a dog?

~~It is an animal~~

↓ It is a 4-legged animal / carnivore

semantic (meaning) property.

4) Syntax Analysis :

→ Performs parsing
→ Check whether a given sentence S belongs to a language L defined by a grammar G or not.

* Grammar :

It is a 4 tuple that consists of

<i> a start symbol
<ii> a set of production rules of the form
$\alpha \rightarrow \beta$
<iii> a set of non-terminals (uppercase letters)
<iv> a set of terminals (lowercase letters).

Rules : $\alpha \rightarrow \beta$
α ↓ non-terminal
β → terminal / non-terminal / both

We check whether $S \in L(G)$ or not using parse tree (leftmost or rightmost derivation).

(Eg):

$L(G)$: $S \rightarrow NP\ VP$

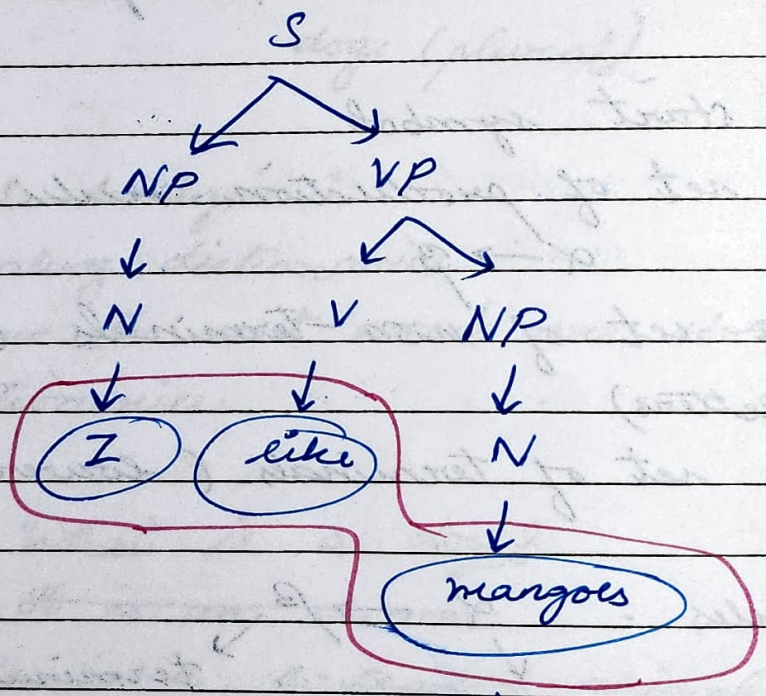
$NP \rightarrow N\ \ VP$

$VP \rightarrow V\ \ NP$

I like Mangoes

Parse tree:

Method 1:



we derived the sentence

$\Rightarrow$ it belongs to the grammar $L(G)$.

1. Check whether the string aabc belongs to L(G) or not

$G: \quad S \rightarrow aAc \quad (R_1)$
$A \rightarrow aA \quad (R_2)$.
$A \rightarrow b \quad (R_3)$.

Method 2:

Non-terminals = $\{S, A\}$
terminals = $\{a, b, c\}$.

$S \rightarrow aAc \quad (R_1)$
$S \rightarrow aaAc \quad (R_2)$
$S \rightarrow aabc \quad (R_3)$.
$\Downarrow$
So $aabc \in L(G)$

5) Semantic Analysis :

→ meaning of sentences (interpretation).

(Eg): I saw Ram with a book ✓
The right door was opened ✓
I saw Ram with X (not complete)

6) Pragmatics :

→ How the sentence is used.

(Eg): Oldere men and women are moved to safer locations, until the flood is over.

7) Discourse :

→ Connection of sentences

(Eg):
1. It was raining today
2. John will not go to play

⇓

As it was raining today, John will not go to play

*. Applications :

Text based :

1. processing of written Text (from newspapers, emails)
2. finding documents on certain topic from large datasets (finding books of an author from a library)

classmate
Date
Page

3. information extraction

4. translation of documents in many languages
(Eg): manuals

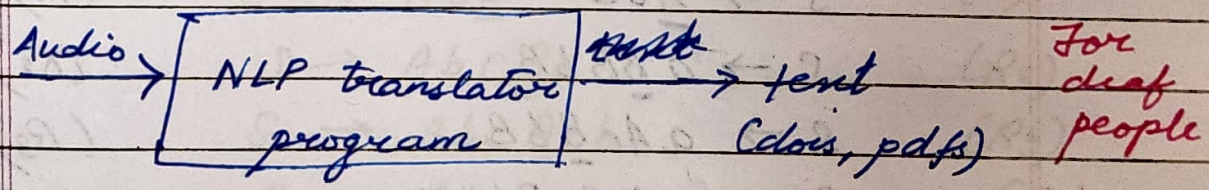
Dialogue based :

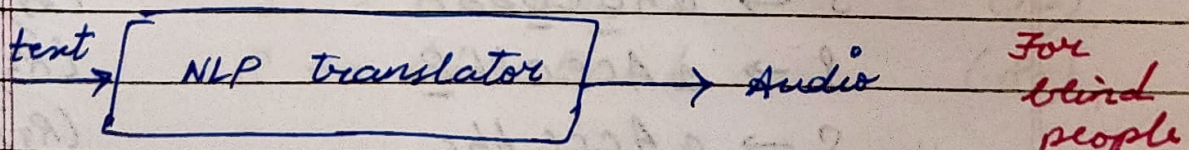
1. human/machine interaction

2. automated tutoring systems

3. question answering system to query a database

Translator programs :





18/08/22

Consider the grammar G defined for a language L

$S \rightarrow AbB \mid bAc$ (R_1)

$A \rightarrow Ab \mid aBB$ (R_2)

$B \rightarrow Ac \mid cBb \mid c$ (R_3)

check whether the following strings $\in L(G)$ or not using leftmost derivation. Construct the parse tree

1. aAcccbbc

$S \rightarrow \underline{A}bB$ (R_1)

$S \rightarrow a\underline{B}BbB$ (R_2)

$S \rightarrow aAc\underline{B}bB$ (R_3)

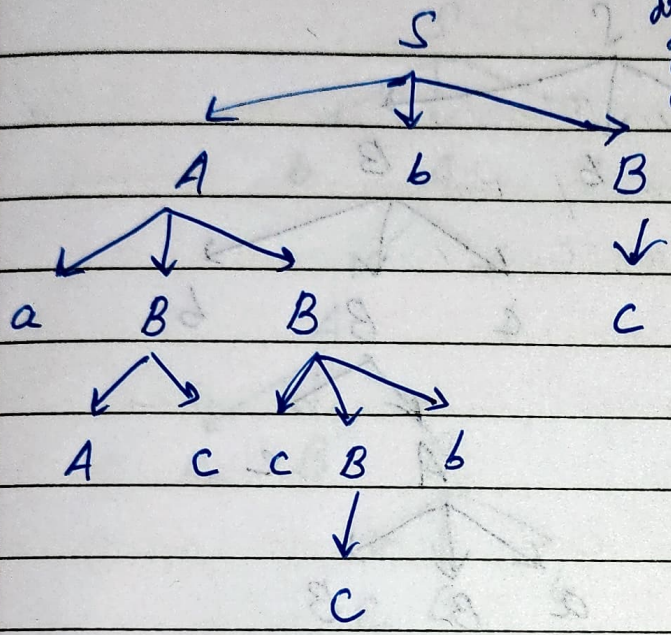
$S \rightarrow aAcc\underline{B}bbB$ (R_2)

$S \rightarrow aAccc\underline{b}bB$ (R_3)

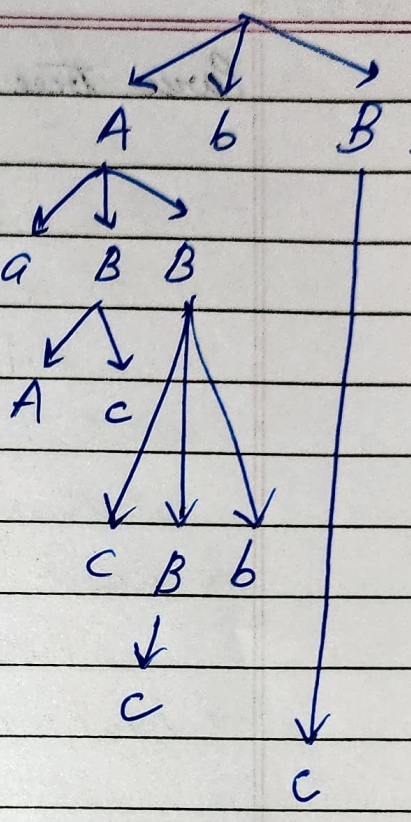
$\therefore S \rightarrow aAcccbbc$ (R_3).

$\therefore \in L(G)$.

Parse tree:



draw at different levels.



2. AbcaBccb.

$S \rightarrow AbB$ (R_1)

$S \rightarrow AbcBb$ (R_3)

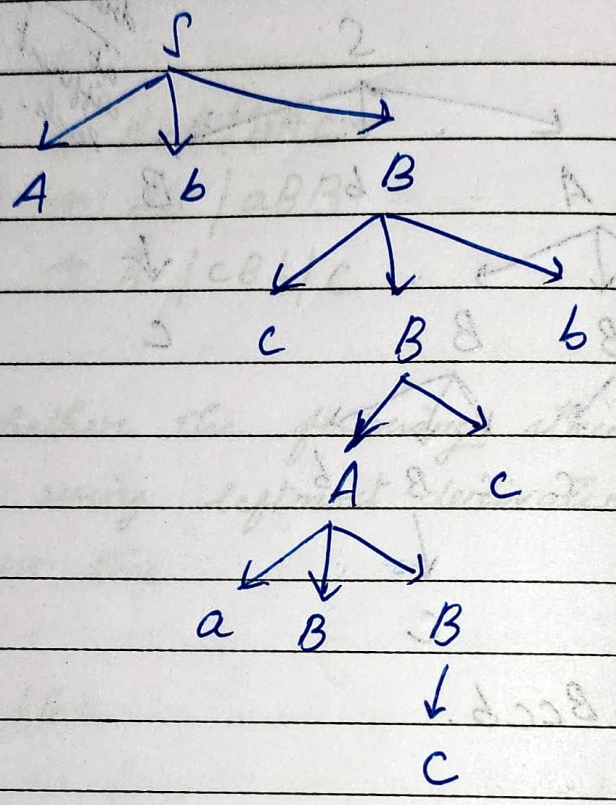
$S \rightarrow AbcAcb$ (R_3)

$S \rightarrow AbcaBBcb$ (R_2).

$S \rightarrow AbcaBccb$ (R_3).

$\in L(G)$.

Parse tree :



3. baBcBbbc

$S \rightarrow bAc$ (R_1).

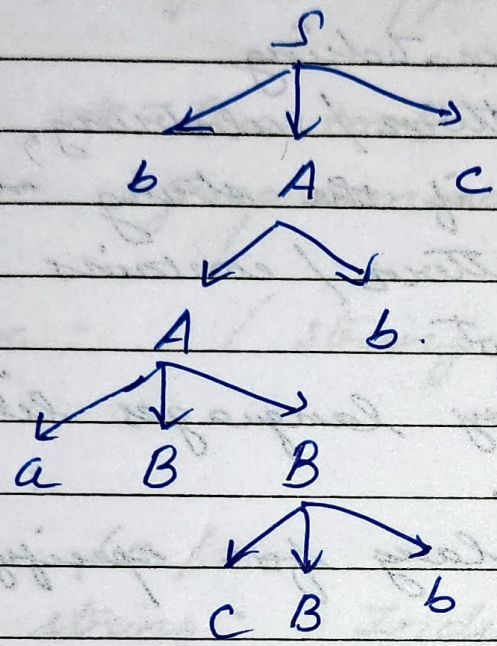
$S \rightarrow bAbc$ (R_2).

$S \rightarrow baBBbc$ (R_2).

$S \rightarrow baBcBbbc$ (R_3).

$\in L(G)$.

Parse tree :



4. baBaBbbc

$S \rightarrow bAc$ (R_1)

~~$S \rightarrow baBBc$~~ ~~(R_2)~~.

~~$S \rightarrow baB$~~

$S \rightarrow bAbc$ (R_2).

$S \rightarrow baBBbc$ (R_2)

$S \rightarrow baBcBbbc$ (R_3)

$\notin L(G)$.

22/08/22

Regular Expression

→ For pattern matching
Given a pattern / substring, RegEx are used to check if the string ~~matches~~ matches the pattern / contains the substring or not
→ Found in many languages like Java, Python, ...
→ RegEx is a lang for specifying token pattern
→ It is an algorithmic notation for characterizing a set of strings.
→ Used to search for a pattern in a given ~~corpus~~ corpus of strings.

(Eg): India (String).

Pattern to be matched : i

0 1 2 3
I n d i a

Found at index : 3

Note: RegEx is case-sensitive
So i ≠ I.

C, C++, Java, Unix, Python, Javascript, Python, Perl support RegEx

RegEx Patterns:

Unix → Lex tool
grep command

Syntax: / string /

Pattern:

Eg: /a/

string: India is a well developed country
↑ ↑
matched

1) brackets: [ ]
↳ disjunction (OR) of characters

(Eg): /[iI]/ ⇒ either i or I

string: India
↑ ↑
matched

2) dash: - => to represent a range of characters.

(Eg): /[A-Z]/ → any char / alphabet from A to Z

/[0-9]/ → any digit

/[a-z]/

3) caret: ^ => to represent negation

/[^A-Z]/ => any string that does not contain the chars btw A to Z

/[^a-z]/ => any string that does not contain a to z

/[^0-9]/ => any string that does not contain numbers

4) Period: . => to represent any character

(Eg): /./ => any characters in the string

/beg.n/ => any character that lies btw the chars/string 'beg' and 'n'

(Eg): begin
↳ .

5) /^ ___ / ⇒ to represent start of string
4 some string

(Eg): /^The/ ⇒ The first word must be 'The'

6) /$/ ⇒ End of string

(Eg): /end$/ ⇒ string ends with the word 'end'.

Occurences :

1) + → one/more occurences of the preceding char/expression

2) * → Zero/more occurences of the ~~following~~ preceding char/expression

3) ? → Zero (or) ~~more~~ one occurences of the preceding char/expression

(Eg): /[a]+/ ⇒ a, aa, aaa, ...

/[ab]+/ ⇒ a, b, aa, ab, ba, bb, ...

a{n} ⇒ n as.

/[a]*/ ⇒ ε, a, aa, aaa, ...

/[ab]*/ ⇒ ε, a, b, aa, ab, ba, bb, ...

/colou?r/ ⇒ color, colour
↳ o or ε

4) {n} ⇒ n occurrences of the previous char/expr

5) {n-m} ⇒ n to m occurrences of the prev. char/expr

Write the Regular Expression for

1. Identifier
2. Int literal
3. List

1. /[A-Za-z][A-Za-z0-9_]*/
2. /[-]?[0-9]+/
3.

1. Identifier :

✓ Starts with an alphabet followed by zero or more no. of alphabets or numbers or any special character like underscore.

alphabet → [A-Z, a-z].
digit → [0-9].
special_char → [_].

2. Int. Literal

✓ One or more number of digits

3. List

✓ Group of one or more items

Answer (given by Siri)

Identifier → / letter (letter / digit / special_char)* /
letter → [A-Z a-z].
digit → [0-9]
special_char → [_] underscore
Int. literal → / digit (digit)* / (OR) / (digit)+ /
List → / Identifier [, Identifier] . /

Derive the RE for the foll. patterns:

1. Name - contains only alphabets and '.'
   Not any digits & special chars

2. Age - contains only digits

3. Phone number - in the form
   3 digits - 3 digits - 4 digits

4. E-mail Id - in the form
   username@domainname
   (Eg): ram_1999@gmail.com

username → {alphanumeric, _, .},
domainname → {string.string}.

23/08/22

Morphology :

✓ Used for syntax analysis & parsing

✓ Forms new words using morphemes

Morphemes : A simple segment of the language that satisfies the following 3 criteria

1) It is a Word or a part of a word with meaning

2) Has the relatively same stable meaning in different verbal environment

3) It cannot be divided into meaningful segments without changing its meaning

2 parts of a morpheme

1. Base / stem / lemma
2. Affix (Prefix / infix / suffix)

Morphological Parsing :

✓ Recognizing the morphemes inside a word

(Eg):

foxes → fox (stem)
foxes → es (suffix)

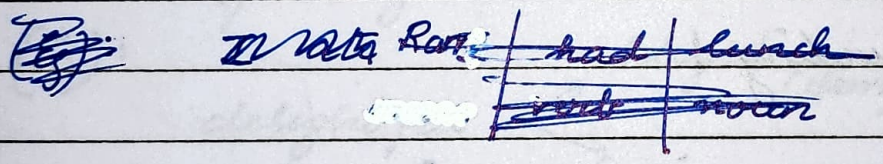
hands → hand (stem)
hands → s (suffix).

Need:

1. to perform spell checking
2. to identify word's POS tags
   ↓
   Part of speech

⇒ used in machine translation
syntactic parsing

3. to identify word's stem → information retrieval.



Formation of new words using Morphemes

1. Inflection:

✓ Combination of stem word with grammatical morphemes

✓ Yields different word class (same?).

(Eg): clean (verb) —ing (suffix)→ cleaning (verb).

2. Derivation :

✓ Similar to inflection but it yields a different word class

(Eg): read (verb) —able (suffix)→ readable (adjective)

delight (verb) —ful (suffix)→ delightful (adj.)

3. Compounding :

✓ Combination of different stem words

(Eg): aqua + phobia → aquaphobia

bi + gram → bigram

4. Cliticization :

✓ Combination of a stem word with clitics.

↳ words having single consonant

(Eg): am, have, will

(Eg): I am → I'm
I have → I've
I will → I'll

Morphological Parser :
↳ Syntax Analyzer

✓ Recognizes morphemes inside a word

Parts :

1. ~~Lexical~~
1. Lexicon
→ Splitting the word into morphemes

(Eg):
delightful → delight (stem), ful (suffix)

2. Morphotics of the language
→ How the morphemes are affixed to the stem.

3. Orthographic rules
→ Spell checking during affixation of the stem words with affixes

(Eg): compile + time → compilation

it is 'a' and not 'e'

read + able → readable

(no error or change in spelling here)

Problems:

1. Productivity:
   → New formations

(Eg): read + able → readable ✓
game + able → gameable ✗

2. False analysis:
   → Analyzing them as words containing wrong suffixes

(Eg): hospitable

3. Baseband

3. Barebound morphemes :

→ Can't stand alone with meaning

base + morpheme → compound
(non-existent) (known)

(Eg): feasible → fease + able
(non-existent) (known)

25/08/22 Language Recognition

✓ Recognizes a language L such that
i) accepts { strings S } $\in L$
ii) rejects { strings S } $\notin L$

✓ ~~Can~~ Implemented using computational devices
i) Finite State ~~Machine (FSM)~~ Automata (FSA)
ii) Finite state transducer (FST)

Finite State Automata

⇒ $S \in L$ or not

✓ defines a formal language L that accepts a set of strings $S \in L$.

✓ It is a directed graph in which
→ nodes are labelled with state names
→ arcs are labelled with symbols that causes transition between the states

Regular,

Reg - noun - singular

cat
dog
flower

Regular - noun - plural

cats
dogs
flowers

} +S

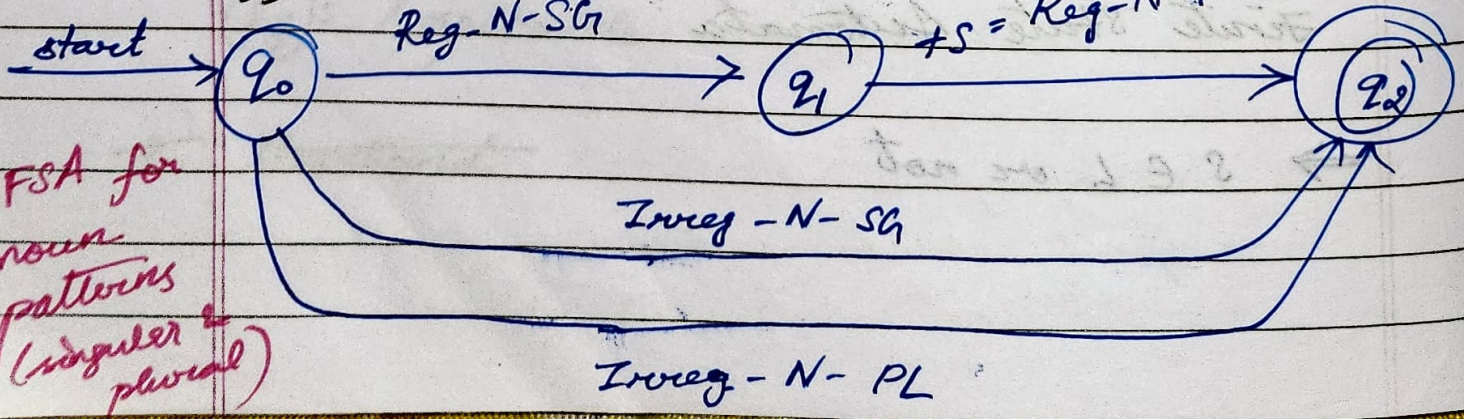
Irreg. - Noun - singular

mouse
ox
fox

Irreg. - noun - plural

mice
oxen
foxes.

FSA:



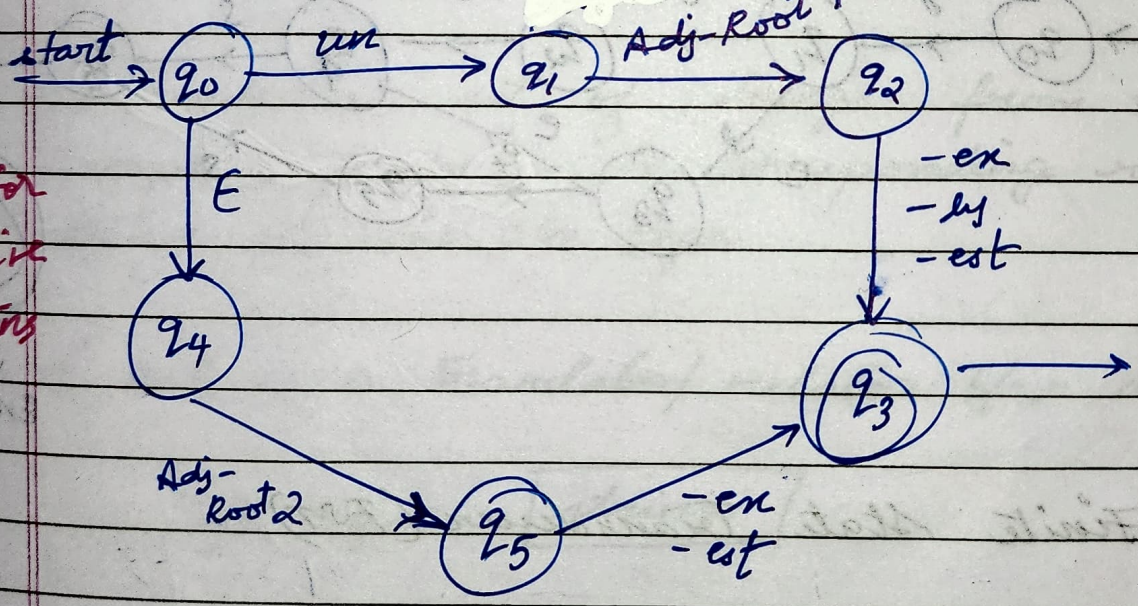
FSA for noun patterns (singular & plural)

Adjectives

Type	Properties	Examples
Adj-Root1	begin with un (or) end with er, ly, est	unhappy, bigger, simply, smallest.
Adj-Root2	does not contain un, ly	bigger, smallest.

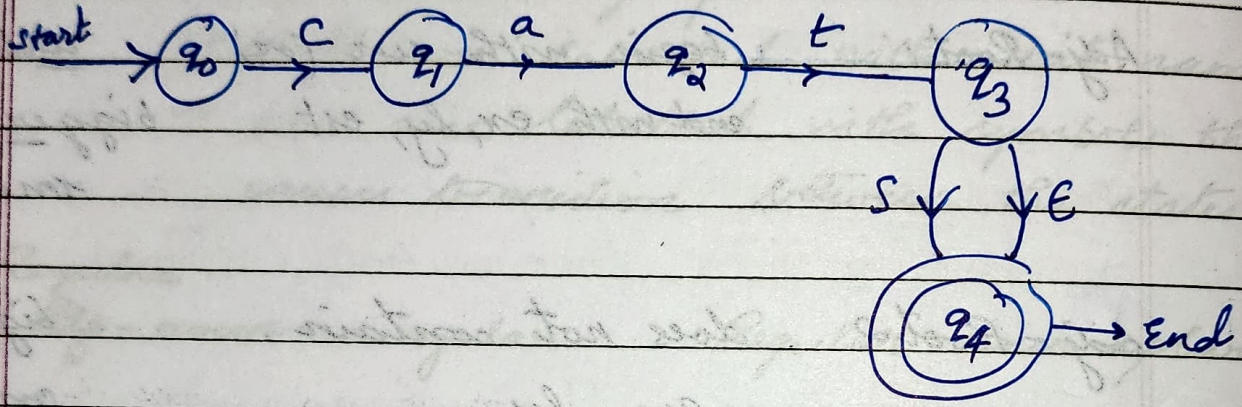
FSA:

FSA for adjective patterns

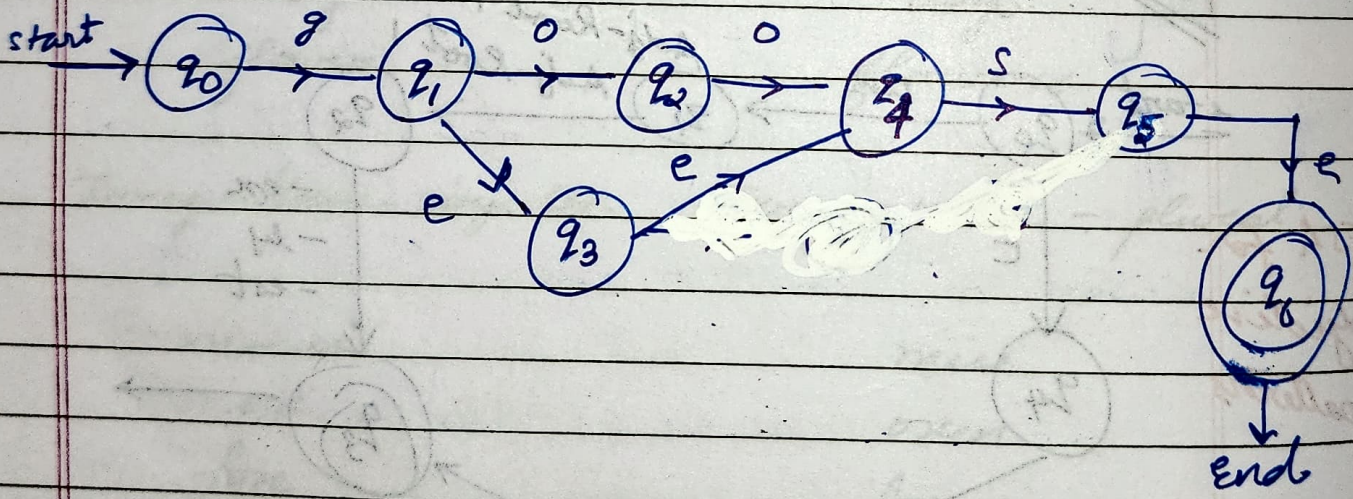


Draw the FSA for the following nouns

1. cat / cats



2. goose / geese



Finite State Transducer (FST)

→ more general form of FSA

→ has 2 tapes → 1 for i/p (contains/stores the i/p string) and the other for o/p (contains the relation btw the morphemes / strings)

Limitations of FSA :

1. Checks whether a set of strings $S \in L$ or not. It does not bother about the relation ~~ships~~ btw the strings

2. Large FSA for entire English words (language?)

✓ FST is a finite state machine with 2 memory tapes – i/p tape and o/p tape

✓ FST – more general than FSA

✓ Reads a set of strings from the input tape & produces the corresponding relation b/w strings on the o/p tape

✓ acts as a translator/relator b/w strings

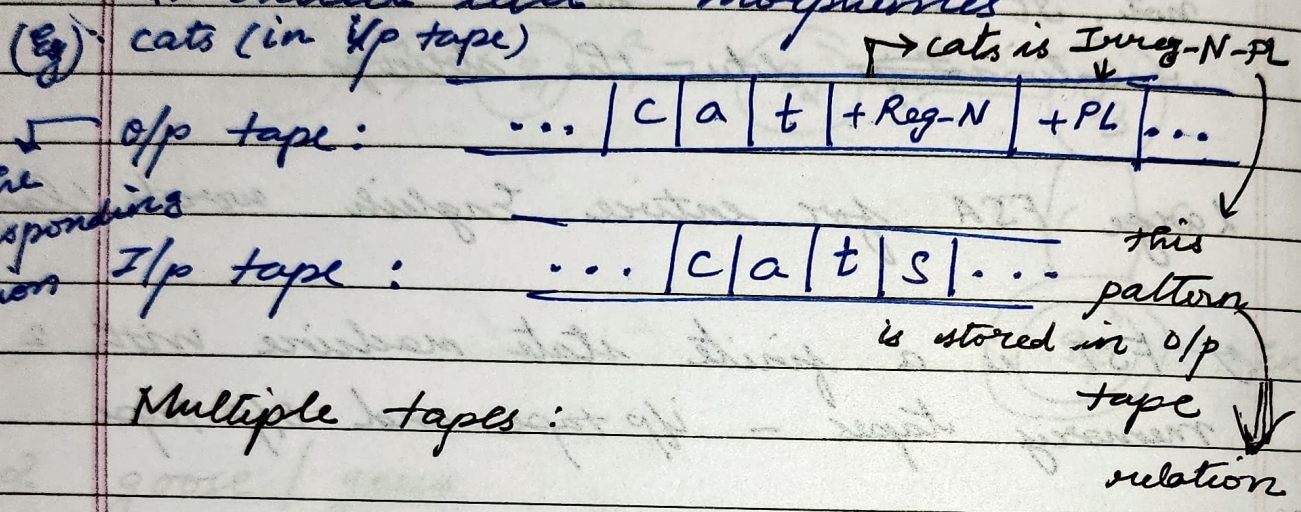
I/p tape: [strings]

O/p tape: [morphemes]

Kimmo - Keisinnimmi's 2-level morphology

1. Surface level - orthographic word
2. Lexical level - morphemes

(Eg): cats (in I/p tape)

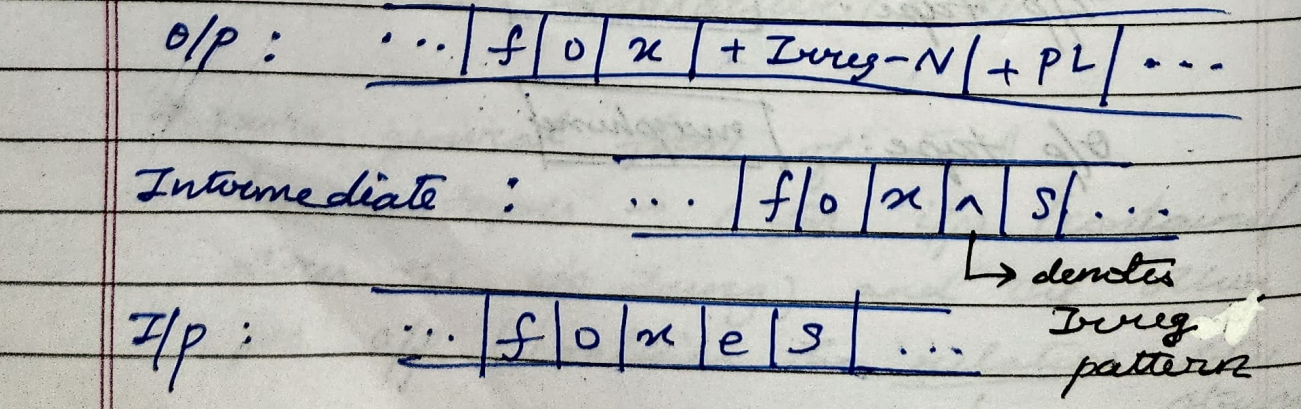


Multiple tapes:

✓ to identify the irregular patterns

✓ intermediate tape is added b/w input & output tapes → for spell check of irregular patterns & add the corresponding symbols

(Eg): Foxes



aBa

Applications of FST:

1. Morphological Analysis
2. Linguistic Analysis
3. Spell check
4. IR

29/08/22

1) Write the RegEx for

a) set of all alphabetic strings ending in a and b

/[A-Za-z]*[ab]$/

b) set of all strings with 2 consecutive repeated words

NA

c) set of all strings from the alphabet a, b such that each a is immediately preceded & and immediately followed by a, b

d) all strings which start at the beginning of the line with an integer and which end at the end of the line with a word

/ [+-]?[0-9]+[_]*[A-Za-z]+ /

e) pattern which places first word of an English sentence in a register. Deal with punctuation.

/^[A-Za-z]+[,:;!?.] /

2) Design an FSA that recognizes simple data expressions like March 15, the 22nd of November, christmas. Try to include such absolute dates. Each edge of the graph should have a word or set of words on it

3) Design an FSA to handle deictic expressions like yesterday, tomorrow, a week from tomorrow, the day before yesterday, the day after tomorrow, sunday, next Tuesday three weeks from saturday

4) Write an FSA for time-of-day expressions like eleven o'clock, twelve-thirty, mid-night, a quarter-to-ten.

5) Draw the FSA and FST for the following patterns

1. rat — RNS
2. rats — RNP
3. flower — RNS
4. flowers — RNP
5. Mango — IoNS (include intermediate tapes in FST)

6.	mangoes	IrNP
7.	orange	RNS
8.	oranges	RNP
9.	ox	IrNS
10.	oxen	IrNP
11.	goose	IrNS
12.	geese	IrNP

30/08/22

Word classes
Parts of a speech (PoS)

→ PoS / Word class (WC) /
~~Language tags~~

2 classes :

1. open class - Eg: nouns, verbs
(can keep creating new nouns and adding on to the existing set)

2. closed class - (Eg): prepositions :

✓ Words are divided into equivalent classes called PoS / Word classes / Morphological class / lexical tags

✓ Provides significant information about the word and its neighbour

Types :

1. Open class :

Doesn't have a fixed membership

(Eg) : new nouns and verbs are borrowed from other languages

- Nouns
- Verbs
- Adjectives
- Adverbs

2. closed class :

Has a fixed membership

(Eg) : Limited prepositions for a given language

- Prepositions
- Determiners
- Pronouns < Personal / Possessive
- Conjunctions

- Auxillary verbs
- Particles (Participle?).
- Numerals.

Open class type

*. Noun:

✓ Name of a person / place / thing / entity

3 types:

i) Proper noun - Name of a specific person / entity / thing
(Eg): Mahatma Gandhi, IBM, Ram, ...

ii) Count nouns - can be counted
occurs in singular / plural
(Eg): goat, goats

iii) Mass nouns - can't be counted
refers to a homogenous group
(Eg): snow, ~~salt~~ salt, stars.

*. Verb :

✓ Represent the action or process

(Eg): draw, go, provide

Ram goes to the Temple everyday.

Ram → Noun
goes → verb

*. Adjectives :

✓ Properties

✓ Many languages have adjectives
for color - black, white
for age - young, old
for value - good, bad, high, low

*. Adverbs :

types :

i) directional / locative

✓ represents the direction or location where the action takes place

(Eg): downhill, here, there

ii) temporal :

✓ time of action / process

(Eg): today, yesterday, now, Monday

iii) degree

✓ to which extent the action takes place

(Eg): extremely, very

iv) manner

✓ manner of action

(Eg): slowly, delicately.

closed class type

*. Prepositions :

✓ on, under, with, to

(Eg): The book is on the table

*. Determiners :

✓ a, an, the

(Eg): The tree is full fruits

*. Pronouns

✓ short cut form of noun phrase

i) Personal pronouns
- I, me, you, it, he, she

ii) Possessive pronouns
- him, his, her, its, mine

iii) Wh - pronouns
- used to frame wh - questions

(Eg): who, whom, whoever, what, ...

what is your name?

*. Conjunctions

✓ to connect two sentences

(Eg): and, because, so

Today is ~~rain~~ raining so Ram can't play outside

* Auxillary verbs :

✓ represent the tense of actions (simple, past, future)

(Eg): may, will

*. Numerals

✓ one, two, three, ...

*. Particles ?

✓ resemble a proposition or adverb

✓ join with a verb to form phrasal verbs.

Examples :

Propositions → on, under, over, near, by, at, from, to, with

Determiners → a, an, the

Personal pronouns → she, he, you, I.

it, me

Possessive pronouns - my, your, his, her, our, their

~~Conjunctions~~

Wh-pronouns - who, whom, what, whatever

Conjunctions - and, but, or, as, if, when, because

Auxillary verbs - can, may, should, are

Particles - up, down, ~~on~~ on, off, in, out, at, by

1/09/22

Tagsets for English Corpus.

3 mostly used tagsets

1. Penn Tree Bank Tagset

   ✓ small size – 45 tagsets.

2. C5 tagset

   ✓ medium size – 67 tagsets
   ✓ to tag British Nation Corpus (BNC)

3. C7 tagset

   ✓ large size – 146 tagsets

Penn Tree Bank PoS tags

Tag	Description	Example
CC	Conjunction	and, but
DT	Determiner	a, the, that
IN	Preposition	of, in, by
JJ	Adjective	

JJR	Adjective comparative
JJS	Adjective superlative
NN	Noun - singular
NNS	Noun - plural
NNP	Proper noun - singular
NNPS	Prop. noun - plural
PP	Personal pronoun
PP$	possessive pronoun
RB	Adverb
RBR	Adverb - comparative
RBS	Adverb - superlative
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund
VBN	verb, past participle
~~VBZ~~ VBZ	verb - 3 sg. pres (3rd person singular present)
WDT	wh - determiner
WP	wh - pronoun
WRB	wh - adverb
(	left paranthesis [, (, {.

PoS Tagging

(Eg): Book that flight

↑ ↑ ↑

VB DT NN

verb determiner noun-singular

PoS tags

→ assigning the proper PoS tag to each word in the sentence

I/p : a sentence / a ~~set~~ string of words and a target of specific words

O/p : a single best tag for each word

(Eg):

I/p : Book that flight

O/p : Book that flight

↑ ↑ ↑

VB DT NN

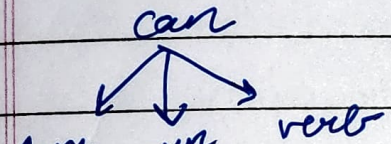
PoS tags (based on Penn Tree Bank target)

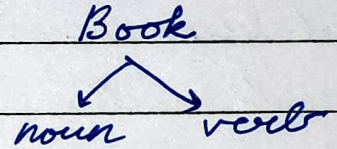
Ambiguity Problem

→ most of the English words are ambiguous

↓

the word can assign more than one tag





(Eg): Book that flight - V

The book is on the table

N

Assign the PoS tags for the following sentences based on Penn Tree Bank Tagset.

1. Does that flight serve dinner?
2. This crap game is over a garage in ~~fifty~~ Fifty-second Street.
3. I need a flight from India
4. I have a friend living in Australia
5. What flights do you have from India

to America?

6. Can you list the non-stop afternoon flights to England?

PoS-tagging algorithms

→ 2 classes

1. Rule-based tagging :

✓ Suitable for larger databases
✓ hand written unambiguous rules for each word

(Eg): specify a word as a noun rather than as an adverb if it follows a determiner

2. Stochastic tagging :

✓ Probabilistic model
→ Probability is assigned to each word with a given tag defined in the given content.

(Eg): Hidden Markov Model (HMM) tagger / Maximum likelihood tagger

05/09/22

Rule based PoS

✓ Suitable for large databases

✓ list of hand-written disambiguous rules

✓ Earlier algorithms - use 2 stage model

First stage:

→ Consists of a dictionary where each word assigns a list of potential PoS.

second stage:

→ Consists of a list of disambiguation that assigns each word with a single part-of-speech

(Eg: ENGTOWL tagger [Voutilainen -1995].

1. each word is run through two-level lexicon transducer where all the possible PoS of each word are returned.

2. a set of constraints are applied to the input sentence to eliminate wrong PoS assigned to each word.

(Eg): one constraint that eliminates all readings of that except the ADV sense
↳ adverbial (only that as an adverb is retained and every other occurrence is eliminated).

Adverbial - that rule

Given input : that

if next word is adj, adverb or quantifier and following which is a sentence boundary and the previous word is not a verb like
then eliminate non-ADV tags
else eliminate ADV tag

1) Apply the ADV-that rule to 'I consider that odd'

Write down the correct PoS for the following sentences

1. I prefer (that) morning flight → apply ADV-that rule
2. Show me the cheapest flight fare
3. Does that flight serve dinner?
4. Which book do you have in that classroom?
5. Which book do you borrow from that library?
6. Can you give some information about that university?

06/09/22

Argmax Computation

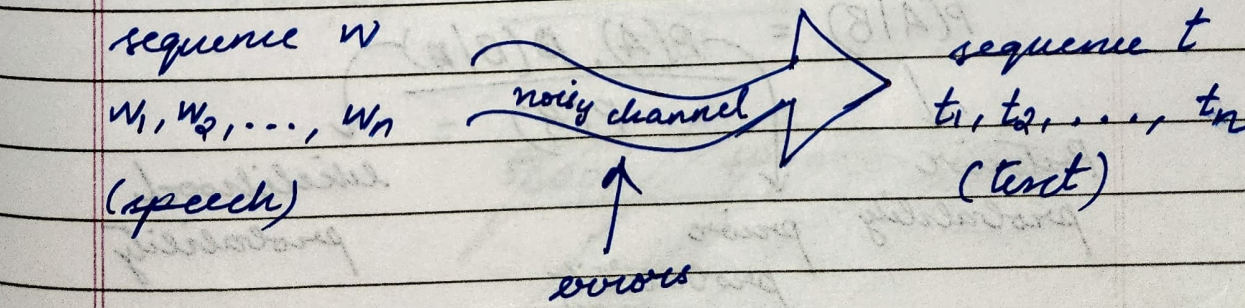
✓ A technique to determine the labels

PoS tags | Named Entity Recognition | Parse trees

✓ This technique can be used to solve the following problems in NLP:

1. PoS tagging
2. Statistical spell check
3. Automated speech recognition
4. Statistical machine translation
5. Probabilistic parsing

Noisy Channel Model



Argmax based computations are based on the noisy channel model

$$w^*/t^* = \underset{w}{\operatorname{argmax}}\ P(w/t).$$

- w^*/t^* : guess at the sequence
- w : correct sequence
- t : transformed text (also consists of noise)

$$y = f(x)$$
$$y^* = f^*(x) \quad (f^* \text{ is max value of } y).$$

$$y^* = \max(y)$$
$$= \max(f(x))$$

$$\therefore x^* = \underset{x}{\operatorname{argmax}}\ f(x)$$

⇓

Find x that maximizes $f(x)$

↓

use Bayes theorem

Given the random values A, B

$$P(A/B) = \frac{P(A).P(B/A)}{P(B)}$$

- $P(A/B)$: Posterior probability
- $P(A)$: prior probability
- $P(B/A)$: likelihood probability

(Eg): It is known that in a population, 1 in 50k has meningitis and 1 in 20 has stiff neck. It is also observed that 50% of meningitis patients have stiff neck & doctor observes that a patient has stiff neck, what is the probability that the patient has meningitis?

Let M be the event that a person has meningitis and S be the event that a person has stiffneck

$$P(M) = \frac{1}{50000}$$

$$P(S) = \frac{1}{20}$$

$$P(S|M) = \frac{1}{2}$$

$$P(M|S) = \frac{P(M) \cdot P(S|M)}{P(S)}$$

$$= \frac{\frac{1}{50000} \times \frac{1}{2}}{\frac{1}{20}}$$

$$= \frac{1}{5}$$

$$= \frac{\frac{1}{50000} \times \frac{1}{2}}{\frac{1}{20}}$$

$$= \frac{1}{5000}$$

$P(M)$ = prior probability of meningitis

$= \frac{1}{50000}$

$P(S)$ = prior probability of stiff neck

$= \frac{1}{20}$

~~$P(M|S) = P($~~

$P(S|M)$ = likelihood probability of stiffneck given meningitis

$= \frac{1}{2}$ 50%

$= 1/2$

$$P(M|S) = \frac{P(M) \cdot P(S|M)}{P(S)}$$

$$= \frac{1/50000 \times 1/2}{1/20} = \frac{1}{5000}$$

Bayes theorem decision principle

✓ Decide in the favour of the value of the variable which has the highest probability among other values of the variables

i.e choose the value which has the highest probability as a decision value

1. PoS tagging

Best tag sequence $t^* = \underset{t}{\text{argmax}}\ P(t/w)$.

↓ each word is assigned the correct tag

$= \underset{t}{\text{argmax}}\ \frac{P(t) \cdot P(w/t)}{P(w)}$

(bcos $P(w)$ is independent of t)

$= \underset{t}{\text{argmax}}\ P(t)\ P(w/t)$

prior probability acts as a filter to remove bad tags

2. Spell checking

$$w^* = \underset{w}{\text{argmax}}\ P(w|t)$$

w^* ↓ best word sequence corresponding to the misspelt word T

$$= \underset{w}{\text{argmax}}\ \frac{P(w) \cdot P(T|w)}{P(T)}$$

$$= \underset{w}{\text{argmax}}\ P(w)\, P(T|w)$$

(Eg): (apple) w

(appee) T

Confusion matrices

1. sub(x, y) - #times y is written for x (y is substituted).
2. Ins(x, y) - #times x is written as xy (y is inserted).
3. Deletion(x, y) - #times xy is written as x (y is deleted).
4. transpose(x, y) - #times xy is written as yx

(Eg):

	(correct word) W	(misspelt word) T	
1.	Appde	Aopde	x=p, y=o sub(x,y).
2.	Fight	Figbht	x=g, y=b Ins(x,y).
3.	Apple	Appe	x=p, y=l deletion(x,y)
4.	Apple	Aplpe	x=p, y=l transpose(x,y).

✓ They are the data structures / matrices that leads to 4 types of misspellings.

Consider the following misspelt words and find out the corresponding confusion matrices

1. abandonned
2. aberation
3. abilties

4. abondon

5. abondoned

6. aborigena

08/09/22

Speech Recognition

1. Automatic Speech Recognition (ASR)
   (~~mapp~~ Maps ~~acoustic~~ speech to
   → Maps the source acoustic (sound) signal into strings of words

2. Automatic Speech Understanding (ASU)
   → complete understanding of the sentence produced

Applications:

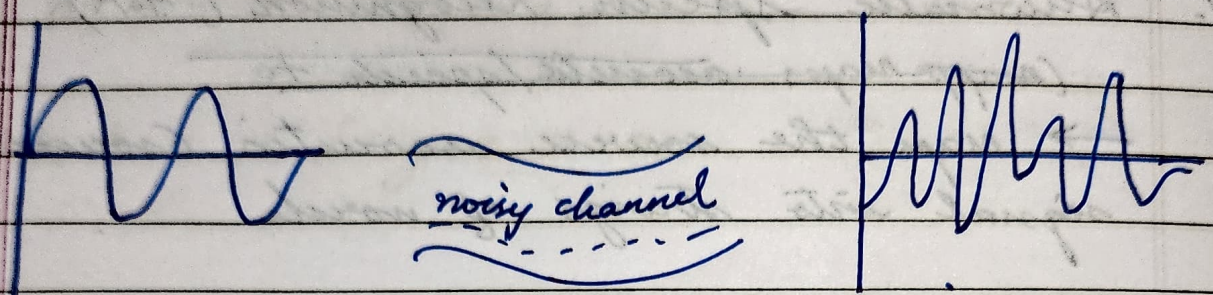
1. HCI (Human Computer Interaction)

2. Telephonic

3. Large vocabular continuous speech recognition (LVCSR)
   → 5000 to 60000 words all run together without a pause

4. Isolated word speech recognition (ISR)
   → each word is preceded and followed by a pause
   → speaker independent

Noisy channel Architecture



source signal

~~$O(o_1, o_2, \ldots$~~

$O(o_1 o_2 \ldots o_n)$

observation

noisy sentence

DECODER

actual sentence (string of words)

$W(w_1 w_2 \ldots w_n)$

It offers a solution to 2 problems :

1. Pick the sentence that best matches the noisy input
   ⇒ use probabilistic model to select the sentence with highest probability of match

2. develop the algorithm over a large set of English sentences that will not ~~select~~ select all but select the ones that best matches - called as decoding

Decoding algorithms :

1. Viterbi / Dynamic Programming
2. stack / A*

Probabilistic models

that are used to generate noisy sentences :

1. N-gram model
   → expresses the probability of sentences being realized as string of words

2. HMM
   → express the probability of words being realized as string of phones.

3. Gaussian model
   → express the probability of phones being realized as acoustic / spectral ~~features~~ features

Goal:

What is the most likelihood sentence among all sentences $\in L$ that best matches the input i.e which one has the highest probability of ~~go~~ generating the noisy sentence.

The problem is defined as follows:

Input signal : O
output sentence : W

$$\hat{W} = \underset{W \in L}{\operatorname{argmax}}\ P(W|O)$$

$$= \underset{W \in L}{\operatorname{argmax}}\ \frac{P(W) \cdot P(O|W)}{P(O)}$$

↳ occurrence of observation
It doesn't change & so it is omitted

$$\hat{W} = \underset{W \in L}{\operatorname{argmax}}\ P(W) \cdot P(O|W)$$

$P(W)$ ↓ prior probability or language model

$P(O|W)$ ↘ likelihood probability or acoustic model

Architecture of a simple speech recognizer.