Module-2

Word Level Analysis

Morphology Analysis

1. Morphology is a study of word formation

- Some words can be divided into two parts but still have meaning .
  Example: handsome (hand + some)
- Many words have meaning in themselves but some words have meaning only when used with other words. Example: sleeping, colourful
- Some words can stand alone. Example : table, local, pity, tree,etc.
- Words parts must be combined in the correct way and systematically.

Morpheme : A morpheme is the smallest unit of form that has meaning in a given language.

Example : Unlikely (Un - like - ly)--- 3 morpheme.
Receive (Re - cieve), transport (trans - port)--- 2 morphemes.

Division/Classification of morphemes.

1. Lexical and grammatical Morphemes
2. Free and bound Morphemes
3. Inflectional and derivational Morphemes

Lexical and grammatical

- Lexical morpheme has meaning in themselves
- Grammatical morphemes have no meaning in themselves they modify the meaning of lexical morphemes
- Noun, adjective, verbs: boy, big, food, girl are lexical morphemes.
- Prepositions, articles, and conjunctions are grammatical morphemes. Example: on, in, an, the, and, or, etc.

Free and bound

- Free morphemes are those which can stand alone as independent words.
- Bound morphemes are those which are used in combination of another morphemes. Example: truth ,sleep are free morpheme and un, fil, ing are bound morpheme.
- Simple word: table, house, etc (single)
- Complex words: player, conceive (two)
- Compound words: girlfriend, football player (two free,etc).

Universal classification of morphemes

- Inflectional morphology
- Derivational morphology

Affixes: affixes are bound morphemes and those morphemes which attach themselves either at the beginning or at the end of the base are called affixes. Example: colourful (base + affix), Dishonest (affix + base), untruthful (affix + base + affix)

Affixes-- prefix (incorrect)
-- suffix (childhood)

Derivational affixes:

- Affixes that are used to derive words i.e. changing word from one type to another are derivational affixes and study of such morphemes is called derivation morphology.
- Derivation morphemes may be either prefix or suffix. Example: disagreement (dis ,agree ,ment).

Inflection affixes

- Inflectional affixes are only suffixes which express grammatical features such as singular/plural,past/present forms of verbs.
- The study of such morphemes is called inflectional morphemes. Examples: bigger (big+er), loves (love+s).

Stemming and Lemmatization

- In natural language processing, there may come a time when you want your program to recognize that the words “ask” and “asked” are just different tenses of the1 same verb. This is the idea of reducing different forms of a word to a core root. Words that are derived from one another can be mapped to a central word or symbol, especially if they have the same core meaning.
- This is where something like stemming or lemmatization comes in.

- Both of these techniques tackle the same idea: Reduce a word to its root or base unit.
- Example : Affectation , affects, affections , affected , affection, affecting -> all of these came from single root work affect.

Stemming

- Stemming algorithms work by cutting off the end or the beginning of the word, taking into account a list of common prefixes and suffixes that can be found in an inflected word.
- This indiscriminate cutting can be successful on some occasions, but not always, and that is why we affirm that this approach presents some limitations.
  (matlab jaroori nahi hai hamesha jo root word ho uss word ko koi meaning bane example : studies ->studi but it should be Study so here lemmatization comes in picture)

Lemmatization

- *Lemmatization* usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the *lemma*
- Example : Lemmatization will map gone,going and went->Go.
- Output of Lemmatization is proper word
- If confronted with the token *saw*, stemming might return just *s*, whereas lemmatization would attempt to return either *see* or *saw* depending on whether the use of the token was as a verb or a noun.

How do they work?

- **Stemming**: there are different algorithms that can be used in the stemming process, but the most common in English is **Porter stemmer**. The rules contained in this algorithm are divided in five different phases numbered from 1 to 5. The purpose of these rules is to reduce the words to the root.
- **Lemmatization**: the key to this methodology is linguistics. To extract the proper lemma, it is necessary to look at the morphological analysis of each word. This requires having dictionaries for every language to provide that kind of analysis.

Regular Expressions

- A regular expression (RE) is a language for specifying text search strings.
- RE helps us to match or find other strings or sets of strings, using a specialized syntax held in a pattern.
- Regular expressions are used to search texts in UNIX as well as in MS WORD in an identical way.
- We have various search engines using a number of RE features.

Properties of Regular Expressions

Followings are some of the important properties of RE –

- American Mathematician Stephen Cole Kleene formalized the Regular Expression language.
- RE is a formula in a special language, which can be used for specifying simple classes of strings, a sequence of symbols. In other words, we can say that RE is an algebraic notation for characterizing a set of strings.
- Regular expression requires two things, one is the pattern that we wish to search and other is a corpus of text from which we need to search.

Mathematically, A Regular Expression can be defined as follows –

- ε is a Regular Expression, which indicates that the language is having an empty string.
- φ is a Regular Expression which denotes that it is an empty language.
- If X and Y are Regular Expressions, then
  - X, Y
  - X.Y(Concatenation of XY)
  - X+Y (Union of X and Y)
  - X*, Y* (Kleen Closure of X and Y) are also regular expressions.
- If a string is derived from the above rules then that would also be a regular expression.

Examples of Regular Expressions

The following table shows a few examples of Regular Expressions –

Regular Expressions	**Regular Set**
(0 + 10*)	{0, 1, 10, 100, 1000, 10000, … }
(0*10*)	{1, 01, 10, 010, 0010, …}
(0 + ε)(1 + ε)	{ε, 0, 1, 01}
(a+b)*	It would be set of strings of a's and b's of any length which also includes the null string i.e. {ε, a, b, aa , ab , bb , ba, aaa…….}
(a+b)*abb	It would be set of strings of a's and b's ending with the string abb i.e. {abb, aabb, babb, aaabb, ababb, …………..}

(11)*	It would be set consisting of even number of 1's which also includes an empty string i.e. {ε, 11, 1111, 111111, ...}
(aa)*(bb)*b	It would be set of strings consisting of even number of a's followed by odd number of b's i.e. {b, aab, aabbb, aabbbbb, aaaab, aaaabbb, ...}
(aa + ab + ba + bb)*	It would be string of a's and b's of even length that can be obtained by concatenating any combination of the strings aa, ab, ba and bb including null i.e. {aa, ab, ba, bb, aaab, aaba, ...}

Regular Sets & Their Properties

It may be defined as the set that represents the value of the regular expression and consists of specific properties.

Properties of regular sets

- If we do the union of two regular sets then the resulting set would also be regula.
- If we do the intersection of two regular sets then the resulting set would also be regular.
- If we do the complement of regular sets, then the resulting set would also be regular.
- If we do the difference of two regular sets, then the resulting set would also be regular.
- If we do the reversal of regular sets, then the resulting set would also be regular.
- If we take the closure of regular sets, then the resulting set would also be regular.
- If we do the concatenation of two regular sets, then the resulting set would also be regular.

Finite State Automata

- The term automata, derived from the Greek word "αὐτόματα" meaning "self-acting", is the plural of automaton which may be defined as an abstract self-propelled computing device that follows a predetermined sequence of operations automatically.
- An automaton having a finite number of states is called a Finite Automaton (FA) or Finite State automata (FSA).

Mathematically, an automaton can be represented by a 5-tuple (Q, Σ, δ, q0, F), where –

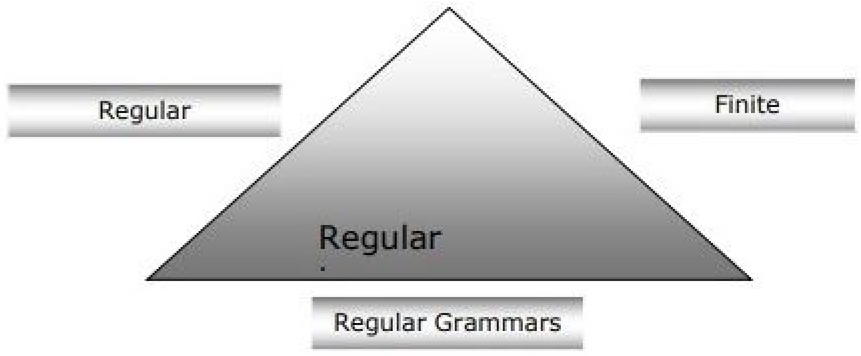
- Q is a finite set of states.
- Σ is a finite set of symbols, called the alphabet of the automaton.
- δ is the transition function
- q0 is the initial state from where any input is processed ($q0 \in Q$).
- F is a set of final state/states of Q ($F \subseteq Q$).

Relation between Finite Automata, Regular Grammars and Regular Expressions

Following points will give us a clear view about the relationship between finite automata, regular grammars and regular expressions –

- As we know that finite state automata are the theoretical foundation of computational work and regular expressions is one way of describing them.
- We can say that any regular expression can be implemented as FSA and any FSA can be described with a regular expression.
- On the other hand, regular expression is a way to characterize a kind of language called regular language. Hence, we can say that regular language can be described with the help of both FSA and regular expression.
- Regular grammar, a formal grammar that can be right-regular or left-regular, is another way to characterize regular language.

Following diagram shows that finite automata, regular expressions and regular grammar are the equivalent ways of describing regular languages.



Types of Finite State Automation (FSA)

Finite state automation is of two types. Let us see what the types are.

Deterministic Finite automation (DFA)

It may be defined as the type of finite automation wherein, for every input symbol we can determine the state to which the machine will move. It has a finite number of states that is why the machine is called Deterministic Finite Automaton (DFA).

Mathematically, a DFA can be represented by a 5-tuple (Q, Σ, δ, q0, F), where –

- Q is a finite set of states.
- Σ is a finite set of symbols, called the alphabet of the automaton.
- δ is the transition function where $\delta: Q \times \Sigma \rightarrow Q$.
- q0 is the initial state from where any input is processed ($q0 \in Q$).
- F is a set of final state/states of Q ($F \subseteq Q$).

Whereas graphically, a DFA can be represented by diagraphs called state diagrams where –

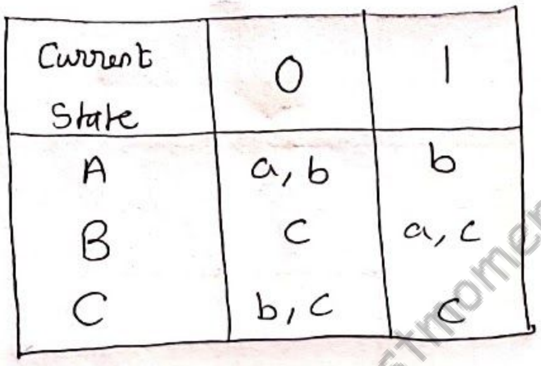
- The states are represented by vertices.
- The transitions are shown by labeled arcs.
- The initial state is represented by an empty incoming arc.
- The final state is represented by a double circle.

Example of DFA

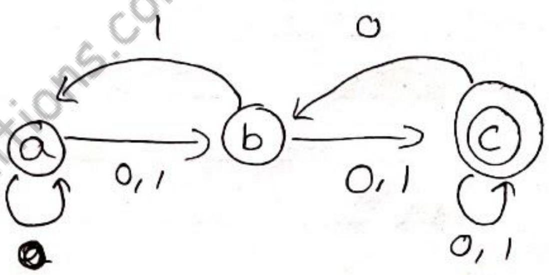
Suppose a DFA be

- Q = {a, b, c},
- Σ = {0, 1},
- q0 = {a},
- F = {c},
- Transition function δ is shown in the table as follows –

Table and diagram



Current State	0	1
A	a, b	b
B	c	a, c
C	b, c	c



Non-deterministic Finite Automation (NDFA)

- It may be defined as the type of finite automation where for every input symbol we cannot determine the state to which the machine will move i.e. the machine can move to any combination of the states.
- It has a finite number of states that is why the machine is called Non-deterministic Finite Automation (NDFA).

Mathematically, NDFA can be represented by a 5-tuple (Q, Σ, δ, q0, F), where –

- Q is a finite set of states.
- Σ is a finite set of symbols, called the alphabet of the automaton.
- δ :-is the transition function where δ: Q × Σ → 2 Q.

- q0 :-is the initial state from where any input is processed (q0 ∈ Q).
- F :-is a set of final state/states of Q (F ⊆ Q).

Whereas graphically (same as DFA), a NDFA can be represented by diagraphs called state diagrams where –

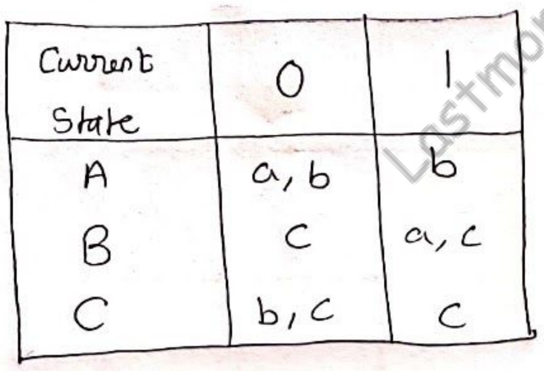
- The states are represented by vertices.
- The transitions are shown by labeled arcs.
- The initial state is represented by an empty incoming arc.
- The final state is represented by double circle.

Example of NDFA

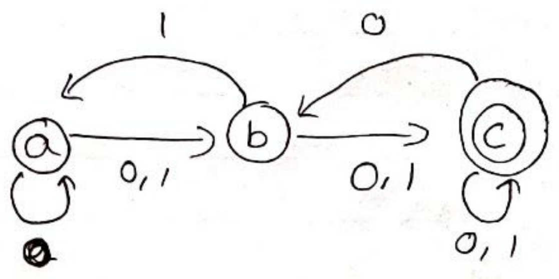
Suppose a NDFA be

- Q = {a, b, c},
- Σ = {0, 1},
- q0 = {a},
- F = {c},
- Transition function δ is shown in the table as follows –

Table and diagram



Current State	0	1
A	a, b	b
B	c	a, c
C	b, c	c



Morphological Parsing

The term morphological parsing is related to the parsing of morphemes. We can define morphological parsing as the problem of recognizing that a word breaks down into smaller meaningful units called morphemes producing some sort of linguistic structure for it. For example, we can break the word *foxes* into two, *fox* and *-es*. We can see that the word *foxes*, is made up of two morphemes, one is *fox* and other is *-es*.

In other sense, we can say that morphology is the study of –

- The formation of words.
- The origin of the words.
- Grammatical forms of the words.
- Use of prefixes and suffixes in the formation of words.
- How parts-of-speech (PoS) of a language are formed.

Types of Morphemes

Morphemes, the smallest meaning-bearing units, can be divided into two types –

- Stems
- Word Order

Stems

It is the core meaningful unit of a word. We can also say that it is the root of the word. For example, in the word foxes, the stem is fox.

- Affixes – As the name suggests, they add some additional meaning and grammatical functions to the words. For example, in the word foxes, the affix is – es.

Further, affixes can also be divided into following four types –

- Prefixes – As the name suggests, prefixes precede the stem. For example, in the word unbuckle, un is the prefix.
- Suffixes – As the name suggests, suffixes follow the stem. For example, in the word cats, -s is the suffix.
- Infixes – As the name suggests, infixes are inserted inside the stem. For example, the word cupful, can be pluralized as cupsful by using -s as the infix.

- Circumfixes – They precede and follow the stem. There are very less examples of circumfixes in English language. A very common example is ‘A-ing’ where we can use -A precede and -ing follows the stem.

Word Order

The order of the words would be decided by morphological parsing. Let us now see the requirements for building a morphological parser –

Lexicon

The very first requirement for building a morphological parser is lexicon, which includes the list of stems and affixes along with the basic information about them. For example, the information like whether the stem is Noun stem or Verb stem, etc.

Morphotactics

It is basically the model of morpheme ordering. In other sense, the model explaining which classes of morphemes can follow other classes of morphemes inside a word. For example, the morphotactic fact is that the English plural morpheme always follows the noun rather than preceding it.

Orthographic rules

These spelling rules are used to model the changes occurring in a word. For example, the rule of converting y to ie in word like city+s = cities not citys.

Language Model/Modeling

- The goal of the language model is to assign probability to a sentence
- There are many sorts of applications for Language Modeling, like: Machine Translation, Spell Correction Speech Recognition, Summarization, Question Answering, Sentiment analysis etc. Each of those tasks require the use of a language *model*.
- Language models are required to represent the text in an understandable form from the machine point of view.
- Language model has alot of application and it helps to choose the right word which fits in sentence and make correct meaning out of it

Some Application of Language Model

Machine translation: translating a sentence saying about height it would probably state that P(tall man)>P(large man)

- P(tall man)>P(large man) as the '*large*' might also refer to weight or general appearance thus, not as probable as '*tall*'

Spelling Correction: Spell correcting sentence: "Put you name into form", so that P(name into **form**)>P(name into **from**)

- P(name into form)>P(name into from)

Speech Recognition: Call my nurse:

P(Call my nurse.)≫P(coal miners)

- Example 2 : I have no idea.

  P(no idea.)≫P(No eye deer.)

  P(no idea.)≫P(No eye deer.)

Summarization, question answering, sentiment analysis etc.

Working of Language Model

Part 1: Defining Language Models

The goal of probabilistic language modelling is to calculate the probability of a sentence of sequence of words:

$$P(W) = P(w_1, w_2, w_3, \ldots, w_n)$$

and can be used to find the probability of the next word in the sequence:

$$P(w_5 | w_1, w_2, w_3, w_4)$$

A model that computes either of these is called a **Language Model**.

Initial Method for Calculating Probabilities

Definition: Conditional Probability

let A and B be two events with P(B) =/= 0, the conditional probability of A given B is:

$$P(A|B) = \frac{P(A, B)}{P(B)}$$

Definition: Chain Rule

In general cases, the formula is as follows:

$$P(x_1, x_2, \ldots, x_n) = P(x_1)P(x_2|x_1)\ldots P(x_n|x_1, \ldots, x_{n-1})$$

The chain rule applied to compute the joined probability of words in a sequence is therefore:

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i | w_1 w_2 \ldots w_{i-1})$$

For example:

$$\begin{gathered} P(\text{" } its\ water\ is\ so\ transparent \text{ "}) = \\ P(its) * \\ P(water|its) * \\ P(is|its\ water) * \\ P(so|its\ water\ is) * \\ P(transparent|its\ water\ is\ so) \end{gathered}$$

This is a lot to calculate, could we not simply estimate this by counting and dividing the results as shown in the following formula:

$$P(transparent|its\ water\ is\ so) = \frac{count(its\ water\ is\ so\ transparent)}{count(its\ water\ is\ so)}$$

In general, no! There are far too many possible sentences in this method that would need to be calculated and we would like to have very sparse data making results unreliable.

Methods using the Markov Assumption

Definition: Markov Property

- *A stochastic process has the Markov property if the conditional probability distribution of future states of the process (conditional on both past and present states) depends only upon the present state, not on the sequence of events that preceded it. A process with this property is called a Markov process.*
- In other words, the probability of the next word can be estimated given only the previous *k* number of words.

For example, if *k=1*:

$$P(transparent|its\ water\ is\ so) \approx P(transparent|so)$$

or if *k=2*:

$$P(transparent|its\ water\ is\ so) \approx P(transparent|is\ so)$$

General equation for the Markov Assumption, *k=i* :

$$P(w_i|w_1 w_2 \ldots w_{i-1}) \approx P(w_i|w_{i-k} \ldots w_{i-1})$$

N-gram Models

From the Markov Assumption, we can formally define N-gram models where *k* = *n-1* as the following:

$$P(w_i|w_1 w_2 \ldots w_{i-1}) \approx P(w_i|w_{i-(n-1)} \ldots w_{i-1})$$

And the simplest versions of this are defined as the Unigram Model (k = 1) and the Bigram Model (k=2).

Unigram Model (k=1):

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_1)$$

Bigram Model (k=2):

$$P(w_i|w_1 w_2 \ldots w_i - 1) \approx P(w_i|w_{i-1})$$

These equations can be extended to compute trigrams, 4-grams, 5-grams, etc. In general, this is an insufficient model of language because **sentences often have long distance dependencies**. For example, the subject of a sentence may be at the start whilst our next word to be predicted occurs mode than 10 words later.

Estimating Bigram Probabilities using the Maximum Likelihood Estimate:

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

Small Example

Given a corpus with the following three sentences, we would like to find the probability that "I" starts the sentence.

<s I am Sam /s>

<s Sam I am /s>

<s I do not like green eggs and ham /s>

where "<s" and "/s>" denote the start and end of the sentence respectively.

Therefore, we have:

$$P(I| < s) = \frac{count(< s, I)}{count(< s)} = \frac{2}{3}$$

In other words, of the three times the sentence started in our corpus, "I" appeared as the first word.