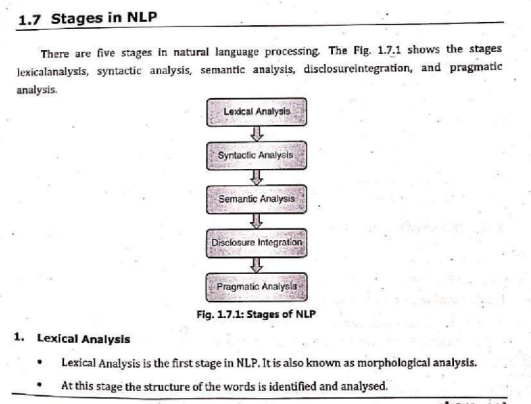
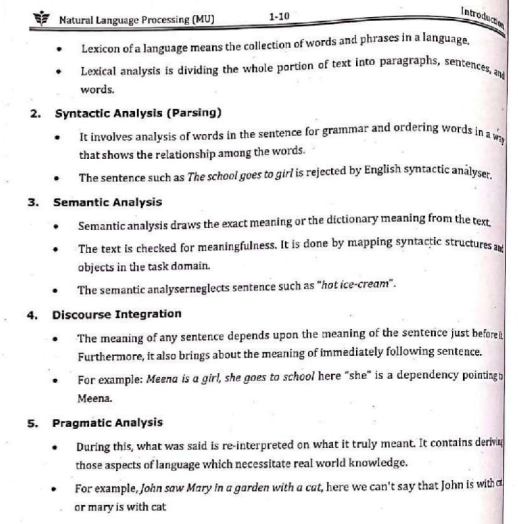
NLP TT-1 Question Bank

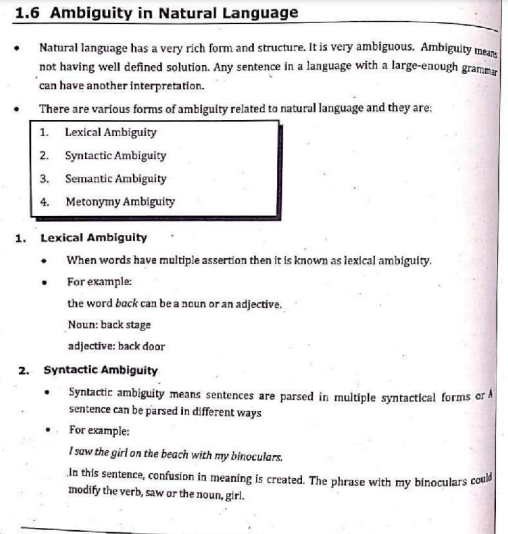
# Module 1:

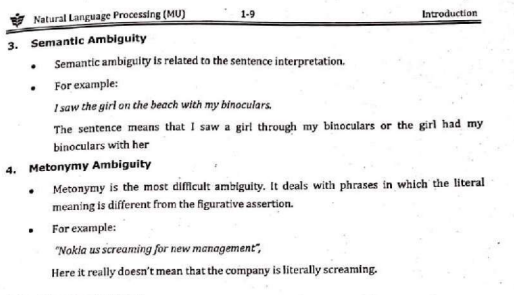
## Stages of NLP





## Ambiguities in NLP





## NLP Pipeline (Explain)

**The NLP Pipeline**

The NLP Pipeline involves the following stages.

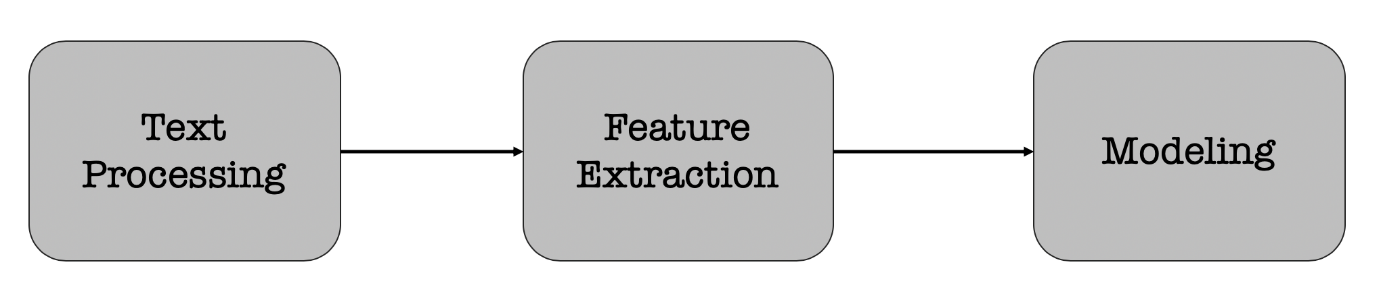
1. Text Processing

* Cleaning
* Normalization
* Tokenization
* Stop Word Removal
* Part of Speech Tagging
* Named Entity Recognition
* Stemming and Lemmatization

1. Feature Extraction

* Bag of Words d
* TF-IDF
* One-hot Encoding
* Word Embeddings

1. Modeling



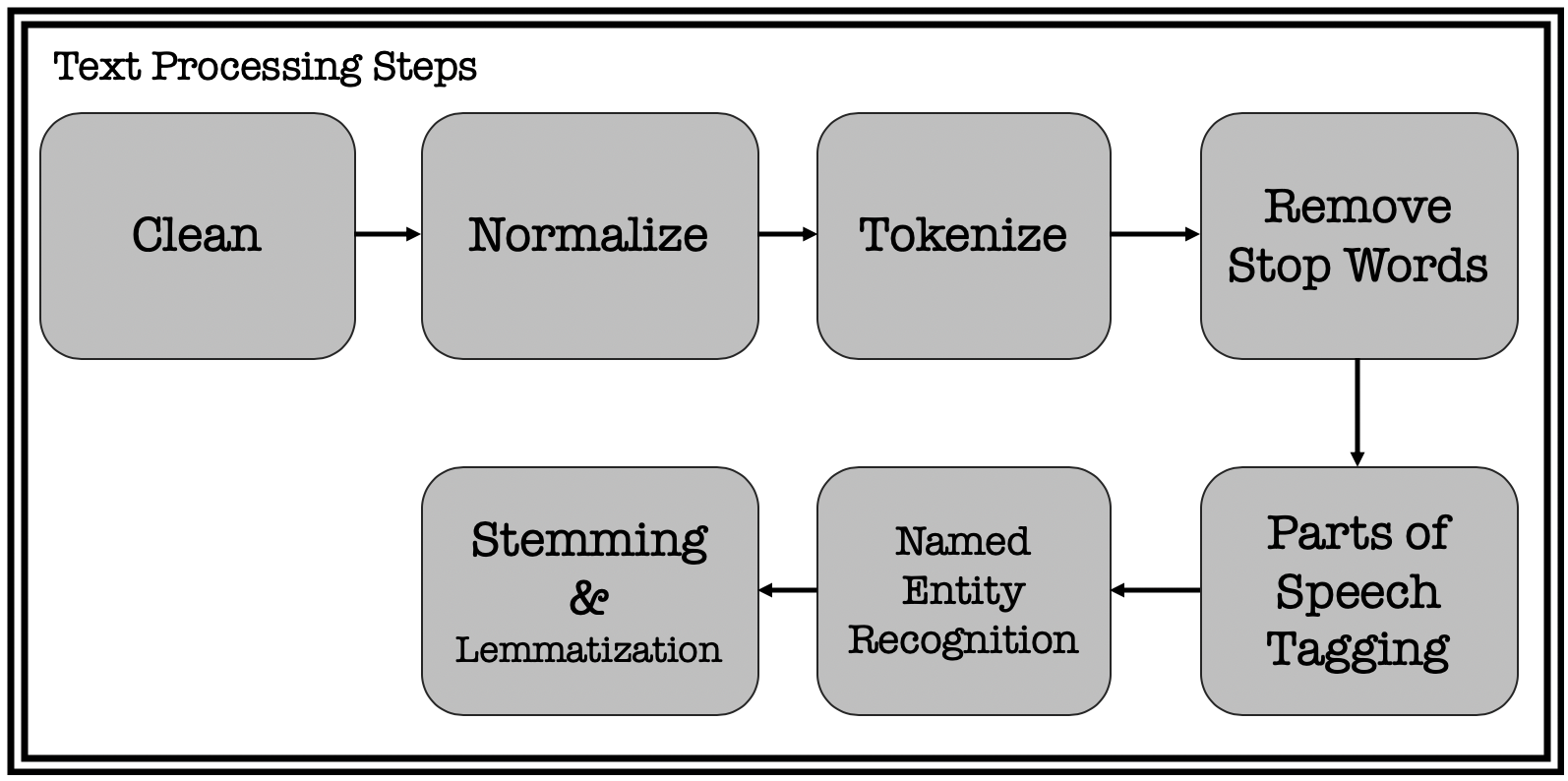
Each stage transforms text in some way and produces an intermediate result that the next stage needs. For example,

* **Text Processing** — take raw input text, clean it, normalize it, and convert it into a form that is suitable for feature extraction.
* **Feature Extraction:** Extract and produce feature representations that are appropriate for the type of NLP task you are trying to accomplish and the type of model you are planning to use.
* **Modeling:** Design a model, fit its parameters to training data, use an optimization procedure, and then use it to make predictions about unseen data.

Text Processing

Text processing is first stage of NLP pipeline that discusses how text data extracted from different sources is prepared for the next stage — **feature extraction**.

* **Cleaning** — The first step in text processingis to clean the data. i.e., removing irrelevant items, such as HTML tags. This can be done in many ways. Example includes using regular expressions, [beautiful soup library](https://www.crummy.com/software/BeautifulSoup/bs4/doc/), CSS selector, etc.
* **Normalization**— The cleaned data is then normalized by converting all words to lowercase and removing punctuation and extra spaces
* **Tokenization** — The normalized data is split into words, also known as tokens
* **Stop Words removal**— After splitting the data into words, the most common words (a, an, the, etc.), also known as stop words are removed
* **Parts of Speech Tagging**— The parts of speech are identified for the remaining words
* **Named Entity Recognition** — The next step is to recognize the named entities in the data
* **Stemming and Lemmatization** — Converting words into their canonical / dictionary forms, using **stemming and lemmatization.**



Steps in Text Processing

\* **Stemming** is a process in which a word is reduced to its stem/root form. i.e., the word running, runs, etc.. can all be reduced to “run”.

\* **Lemmatization** is another technique used to reduce words to a normalized form. In this case, the transformation actually uses a dictionary to map different variants of a word to its root. With this approach, the non-trivial inflections such as is, are, was, were, are mapped back to root ‘be’.

After performing these steps, the text will look very different from the original data, but it captures the essence of what was being conveyed in a form that is easier to work with.

Feature Extraction

Text data is represented on modern computers using an encoding such as ASCII or Unicode that maps every character to a number. Computer stores and transmits these values as binary, zeros and ones, which have an implicit ordering. Individual characters don’t carry much meaning at all and can mislead the NLP algorithms.

**Bag of words (BOW) model**

A bag of words model treats each document as an un-ordered list or bag of words. The word document refers to a unit of text that is being analyzed. For example, while performing a sentiment analysis on tweets, each tweet is considered as a document.

**Term Frequency — Inverse Document Frequency (TF-IDF)**

One limitation of bag of words approach is that it treats every word as being equally important. Whereas, some words occur very frequently in a corpus. Consider a financial document for example. “Cost” or “price” is a very common term.

This limitation can be compensated for by counting number of documents in which each word occurs, known as document frequency, and then dividing the term frequency by document frequency of that term.

This gives us a metric that is proportional to frequency of a term in document, but inversely proportional to number of documents it appears in. This highlights the words that are more unique to a document, thus better for characterizing it.

This approach is called Term Frequency — Inverse Document Frequency (TF-IDF).

One-hot encoding

Another way to represent words is to use one-hot encoding. It’s just like bag of words but only that each word is kept in each bag and a vector is built for it.

**Word Embeddings**

One-hot encoding doesn’t work in every situation. It breaks down when there is a large vocabulary to deal with, because the size of word representation grows with number of words. It is required that word representation is limited to a fixed-size vector.

In other words, an embedding for each word is to be found in vector space that is exhibiting some desired properties. i.e. if two words are similar in meaning, they should be closer to each others compared to the words that are not. And if two pairs of words have similar difference in meanings, they should be approximately equally separated in the embedded space.

This representation can be used for various purposes like finding analogies, synonyms and antonyms, classifying words as positive, negative, neutral, etc.

Modeling

The final stage of the NLP pipeline is **modeling**, which includes designing a statistical or machine learning model, fitting its parameters to training data, using an optimization procedure, and then using it to make predictions about unseen data.

# 

# 

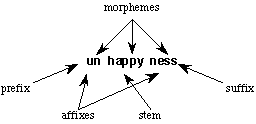
# 

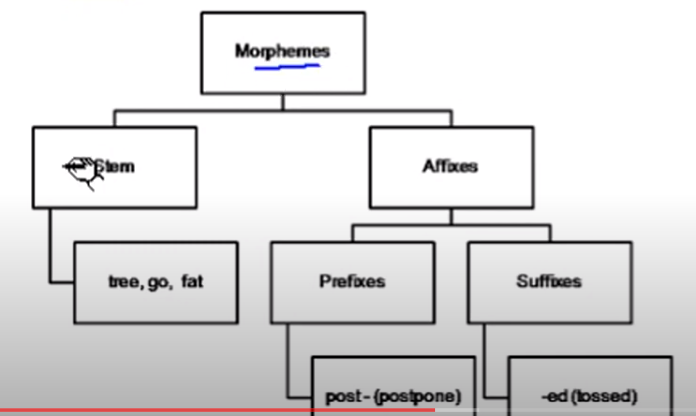
# 

**UNIT-2**

**Word Level Analysis Morphology analysis –survey of English Morphology, Inflectional morphology & Derivational morphology, Lemmatization, Regular expression, finite automata, finite state transducers (FST) ,Morphological parsing with FST , Lexicon free FST Porter stemmer. N –Grams- N-gram language model, N-gram for spelling correction.**

Morphology is the study of the structure and formation of words. Its most important unit is the *morpheme*, which is defined as the "minimal unit of meaning". (Linguistics textbooks usually define it slightly differently as "the minimal unit of grammatical analysis".) Consider a word like: "unhappiness". This has three parts:





There are three morphemes, each carrying a certain amount of meaning. *un* means "not", while *ness* means "being in a state or condition". *Happy* is a *free morpheme* because it can appear on its own (as a "word" in its own right). *Bound morphemes* have to be attached to a free morpheme, and so cannot be words in their own right. Thus you can't have sentences in English such as "Jason feels very un ness today".

**Inflectional ⋅**

An inflectional morpheme is added to a noun, verb, adjective or adverb to assign a particular grammatical property to that word such as: tense, number, possession, or comparison.

Examples of inflectional morphemes are:

Plural: -s, -z, -iz Like in: cats, horses, dogs.

Tense: -d, -t, -id, -ing Like in: stopped, running, stirred, waited

Possession: -‘s Like in: Alex’s

Comparison: -er, -en Like in: greater, heighten \*note that –er is also a derivational morpheme so don’t mix them up!!

⋅ These do do not change the essential meaning or the grammatical category of a word. Adjectives stay adjectives, nouns remain nouns, and verbs stay verbs.

⋅ In English, all inflectional morphemes are suffixes (i.e. they all only attach to the end of words).

⋅ There can only be one inflectional morpheme per word

**Derivational**

⋅ Derivational morphemes tend to change the grammatical category of a word but not always!

⋅ There can be multiple derivational morphemes per word and they can be prefixes, affixes, or suffixes. For example, the word “transformation” contains two derivational morphemes:

trans (prefix) -form (root) -ation (suffix)

Some examples of derivational morphemes are:

* ful like in ‘beautiful’ => beauty (N) + ful (A) = beautiful (A)
* able like in ‘moldable’ => mold (V) + able (A) = moldable (A)
* er like in ‘singer’ => sing (V) + er (N) = singer (N)
* nes like in ‘happiness’ => happy (A) + nes (N) = happiness (N)
* ify like in ‘classify’ => class (N) + ify (V) = classify (V)

Page Break

***Lemmatization:***

***It is the process of converting a word to its base form. The difference between stemming and lemmatization is, lemmatization considers the context and converts the word to its meaningful base form, whereas stemming just removes the last few characters, often leading to incorrect meanings and spelling errors.***

For example, lemmatization would correctly identify the base form of ‘caring’ to ‘care’, whereas, stemming would cut off the ‘ing’ part and convert it to car.

‘Caring’ -> Lemmatization -> ‘Care’

‘Caring’ -> Stemming -> ‘Car’

Also, sometimes, the same word can have multiple different ‘lemma’s. So, based on the context it’s used, you should identify the ‘part-of-speech’ (POS) tag for the word in that specific context and extract the appropriate lemma. Examples of implementing this comes in the following sections.

***Porter Stemming Algorithm:***

In linguistics (study of language and its structure), a **stem** is part of a word, that is common to all of its inflected variants.

* CONNECT
* CONNECTED
* CONNECTION
* CONNECTING

Above words are **inflected variants** of CONNECT. Hence, CONNECT is a stem. To this stem we can add different suffixes to form different words.

The process of reducing such inflected (or sometimes derived) words to their word stem is known as **Stemming**. For example, CONNECTED, CONNECTION and CONNECTING can be reduced to the stem CONNECT.

The **Porter Stemming algorithm** (or **Porter Stemmer**) is used to **remove the suffixes from an English word and obtain its stem** which becomes very useful in the field of **Information Retrieval (IR)**. This process reduces the number of terms kept by an IR system which will be advantageous both in terms of space and time complexity. This algorithm was developed by a British Computer Scientist named **Martin F. Porter**. You can visit the [**official home page**](https://tartarus.org/martin/PorterStemmer/index.html) of the Porter stemming algorithm for further information.

First, a few terms and expressions will be introduced, which will be helpful for the ease of explanation.

**Consonants and Vowels**

A **consonant** is a letter other than the vowels and other than a letter “Y” preceded by a consonant. So in “TOY” the consonants are “T” and “Y”, and in “SYZYGY” they are “S”, “Z” and “G”.

If a letter is not a consonant it is a **vowel**.

A consonant will be denoted by **c** and a vowel by **v**.

A list of one or more consecutive consonants (ccc…) will be denoted by **C**, and a list of one or more consecutive vowels (vvv…) will be denoted by **V**. Any word, or part of a word, therefore has one of the four forms given below.

* **CVCV … C →** collection, management
* **CVCV … V →** conclude, revise
* **VCVC … C →** entertainment, illumination
* **VCVC … V →** illustrate, abundance

All of these forms can be represented using a single form as,

**[C]VCVC … [V]**

Here the square brackets denote arbitrary presence of consonants or vowels.

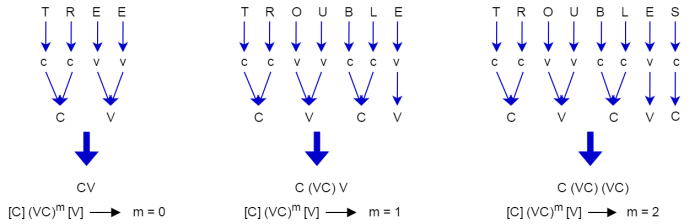
(VC)m denotes VC repeated m times. So the above expression can be written as,

**[C](VC)m[V]**

**What is m?**

The value **m** found in the above expression is called the **measure** of any word or word part when represented in the form **[C](VC)m[V]**. Here are some examples for different values of m:

* m=0 → TREE, TR, EE, Y, BY
* m=1 → TROUBLE, OATS, TREES, IVY
* m=2 → TROUBLES, PRIVATE, OATEN, ROBBERY



**Rules**

The rules for replacing (or removing) a suffix will be given in the form as shown below.

**(condition) S1 → S2**

This means that if a word ends with the suffix S1, and the stem before S1 satisfies the given condition, S1 is replaced by S2. The condition is usually given in terms of m in regard to the stem before S1.

(m > 1) EMENT →

Here S1 is ‘EMENT’ and S2 is null. This would map REPLACEMENT to REPLAC, since REPLAC is a word part for which m = 2.

**Conditions**

The conditions may contain the following:

* \*S – the stem ends with S (and similarly for the other letters)
* \*v\* – the stem contains a vowel
* \*d – the stem ends with a double consonant (e.g. -TT, -SS)
* \*o – the stem ends cvc, where the second c is not W, X or Y (e.g. -WIL, -HOP)

And the condition part may also contain expressions with and, or and not.

(m>1 and (\*S or \*T)) tests for a stem with m>1 ending in S or T.

(\*d and not (\*L or \*S or \*Z)) tests for a stem ending with a double consonant and does not end with letters L, S or Z.

**How are rules obeyed?**

In a set of rules written beneath each other, only one is obeyed, and this will be the one with the longest matching S1 for the given word. For example, with the following rules,

1. SSES → SS
2. IES → I
3. SS → SS
4. S →

(Here the conditions are all null) CARESSES maps to CARESS since SSES is the longest match for S1. Equally CARESS maps to CARESS (since S1=”SS”) and CARES to CARE (since S1=”S”).

**The Algorithm**

**Step 1a**

1. SSES → SS
2. IES → I
3. SS → SS
4. S →

**Step 1b**

1. (m>0) EED → EE
2. (\*v\*) ED →
3. (\*v\*) ING →

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

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

**Step 1c**

1. (\*v\*) Y → I

**Step 2**

1. (m>0) ATIONAL → ATE
2. (m>0) TIONAL → TION
3. (m>0) ENCI → ENCE
4. (m>0) ANCI → ANCE
5. (m>0) IZER → IZE
6. (m>0) ABLI → ABLE
7. (m>0) ALLI → AL
8. (m>0) ENTLI → ENT
9. (m>0) ELI → E
10. (m>0) OUSLI → OUS
11. (m>0) IZATION → IZE
12. (m>0) ATION → ATE
13. (m>0) ATOR → ATE
14. (m>0) ALISM → AL
15. (m>0) IVENESS → IVE
16. (m>0) FULNESS → FUL
17. (m>0) OUSNESS → OUS
18. (m>0) ALITI → AL
19. (m>0) IVITI → IVE
20. (m>0) BILITI → BLE

**Step 3**

1. (m>0) ICATE → IC
2. (m>0) ATIVE →
3. (m>0) ALIZE → AL
4. (m>0) ICITI → IC
5. (m>0) ICAL → IC
6. (m>0) FUL →
7. (m>0) NESS →

**Step 4**

1. (m>1) AL →
2. (m>1) ANCE →
3. (m>1) ENCE →
4. (m>1) ER →
5. (m>1) IC →
6. (m>1) ABLE →
7. (m>1) IBLE →
8. (m>1) ANT →
9. (m>1) EMENT →
10. (m>1) MENT →
11. (m>1) ENT →
12. (m>1 and (\*S or \*T)) ION →
13. (m>1) OU →
14. (m>1) ISM →
15. (m>1) ATE →
16. (m>1) ITI →
17. (m>1) OUS →
18. (m>1) IVE →
19. (m>1) IZE →

**Step 5a**

1. (m>1) E →
2. (m=1 and not \*o) E →

**Step 5b**

1. (m > 1 and \*d and \*L) → single letter

For each word you input to the algorithm, all the steps from 1 to 5 will be executed and the output will be produced at the end.

**Example Inputs**

Let’s consider a few example inputs and check what will be their stem outputs.

**Example 1**

In the first example, we input the word **MULTIDIMENSIONAL** to the Porter Stemming algorithm. Let’s see what happens as the word goes through steps 1 to 5.



* The suffix will not match any of the cases found in steps 1, 2 and 3.
* Then it comes to step 4.
* The stem of the word has m > 1 (since m = 5) and ends with “**AL**”.
* Hence in step 4, “**AL**” is deleted (replaced with null).
* Calling step 5 will not change the stem further.
* Finally the output will be **MULTIDIMENSION**.

MULTIDIMENSIONAL **→ MULTIDIMENSION**

**Example 2**

In the second example, we input the word **CHARACTERIZATION** to the Porter Stemming algorithm. Let’s see what happens as the word goes through steps 1 to 5.



* The suffix will not match any of the cases found in step 1.
* So it will move to step 2.
* The stem of the word has m > 0 (since m = 3) and ends with “**IZATION**”.
* Hence in step 2, “**IZATION**” will be replaced with “**IZE**”.
* Then the new stem will be **CHARACTERIZE**.
* Step 3 will not match any of the suffixes and hence will move to step 4.
* Now m > 1 (since m = 3) and the stem ends with **“IZE”**.
* So in step 4, **“IZE”** will be deleted (replaced with null).
* No change will happen to the stem in other steps.
* Finally the output will be **CHARACTER**.

CHARACTERIZATION → CHARACTERIZE **→ CHARACTER**

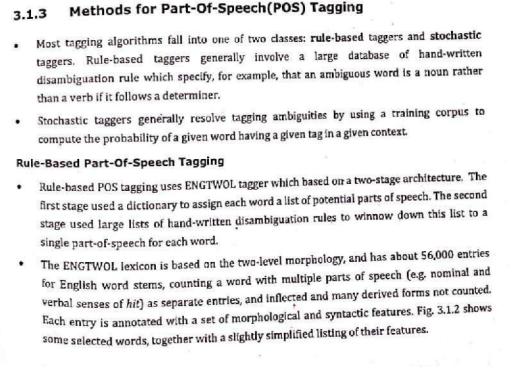
***N-gram Language models:***

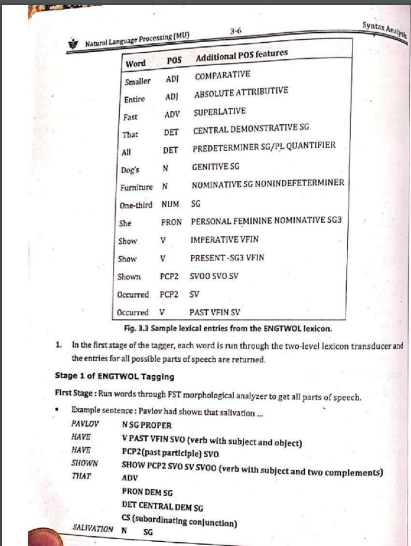
[3.pdf (stanford.edu)](http://web.stanford.edu/~jurafsky/slp3/3.pdf) (text book)

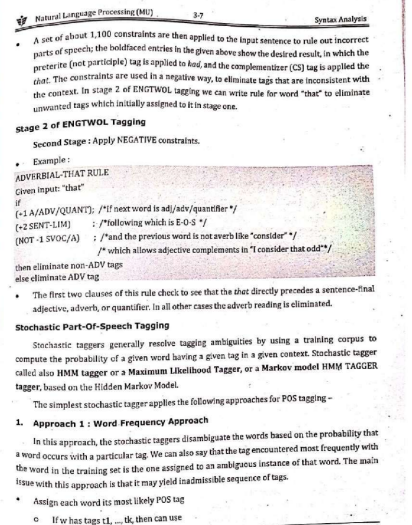
[Ngrams.2013.ppt (syr.edu)](http://classes.ischool.syr.edu/ist664/NLPfall2013/Ngrams.2013.ppt.pdf)(ppt)

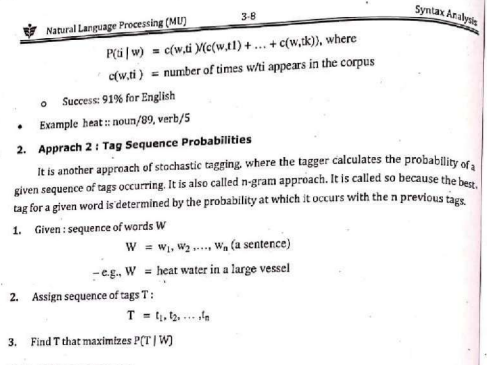
# Module 3

## Types of POS Taggers

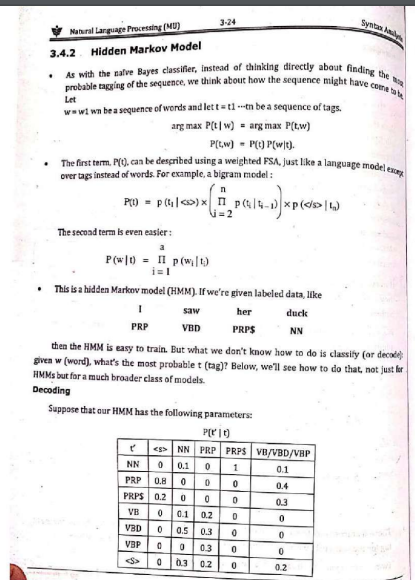


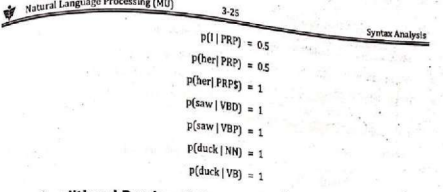




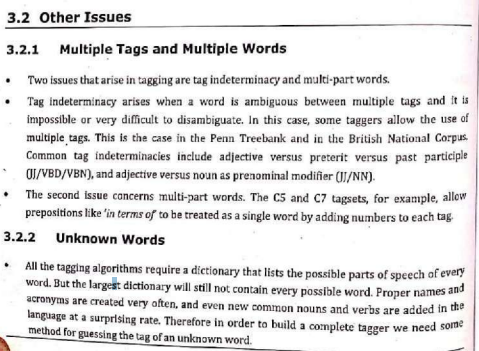


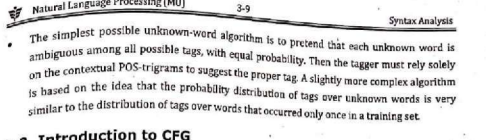
## How do you use HMM in POS Tagging





## Issues with HMM





# Assignment - 1

* 1. Differentiate b/w Interpolation and Backoffx
  2. Viterbi algorithm is a variation of the forward algorithm which considers all words simultaneously in order to compute the most likely path.
  3. Corpus: <s> I am from DJ </s>

<s> I am a teacher </s>

<s> All students are good and intelligent </s>

<s> Students from DJ score high marks </s>

Test Data: <s> students are from DJ </s>

* 1. Corpus: John read Moby Dick

Mary read a different book

She read a book by Cher

Test Data:

* + 1. John read a book
    2. Cher read a book