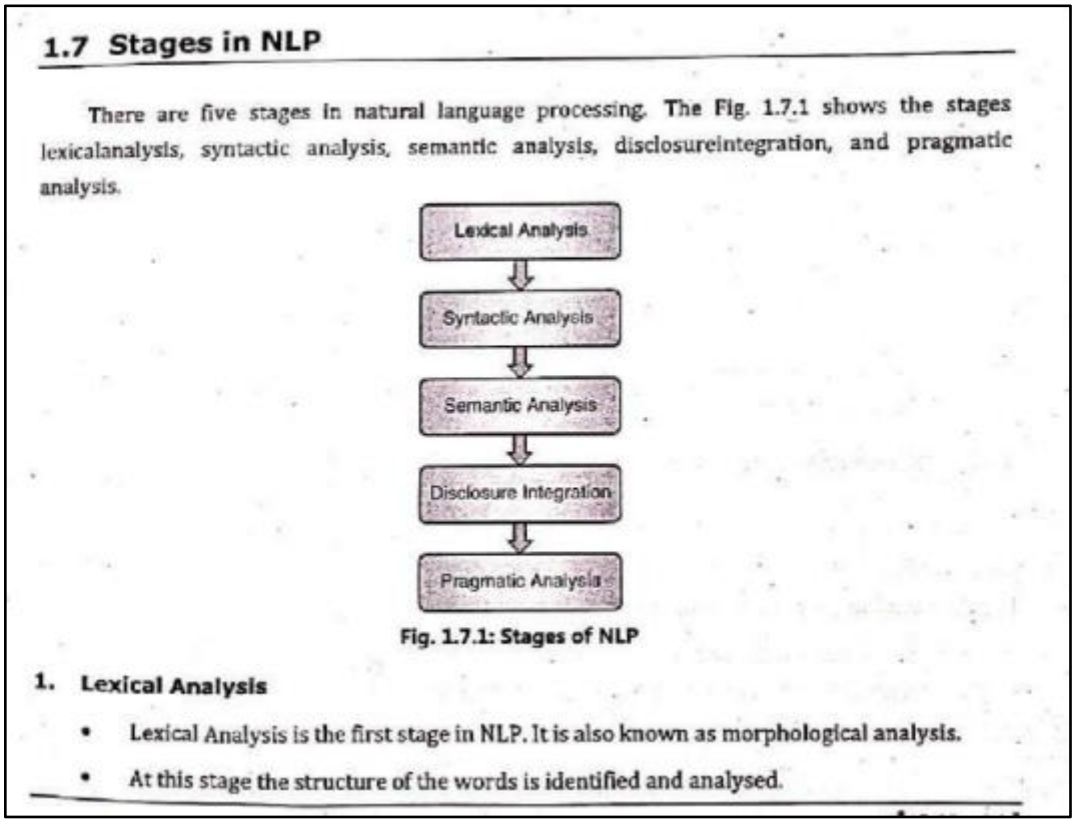
NLP TT-1 Question Bank

Module 1:

1. Stages of NLP



1.7 Stages in NLP

There are five stages in natural language processing. The Fig. 1.7.1 shows the stages lexicalanalysis, syntactic analysis, semantic analysis, disclosureintegration, and pragmatic analysis.

Lexical Analysis
Syntactic Analysis
Semantic Analysis
Disclosure Integration
Pragmatic Analysis

Fig. 1.7.1: Stages of NLP

1. **Lexical Analysis**
   - Lexical Analysis is the first stage in NLP. It is also known as morphological analysis.
   - At this stage the structure of the words is identified and analysed.

- Lexicon of a language means the collection of words and phrases in a language.
- Lexical analysis is dividing the whole portion of text into paragraphs, sentences, and words.

2. Syntactic Analysis (Parsing)

- It involves analysis of words in the sentence for grammar and ordering words in a way that shows the relationship among the words.
- The sentence such as *The school goes to girl* is rejected by English syntactic analyser.

3. Semantic Analysis

- Semantic analysis draws the exact meaning or the dictionary meaning from the text.
- The text is checked for meaningfulness. It is done by mapping syntactic structures and objects in the task domain.
- The semantic analyserneglects sentence such as "*hot ice-cream*".

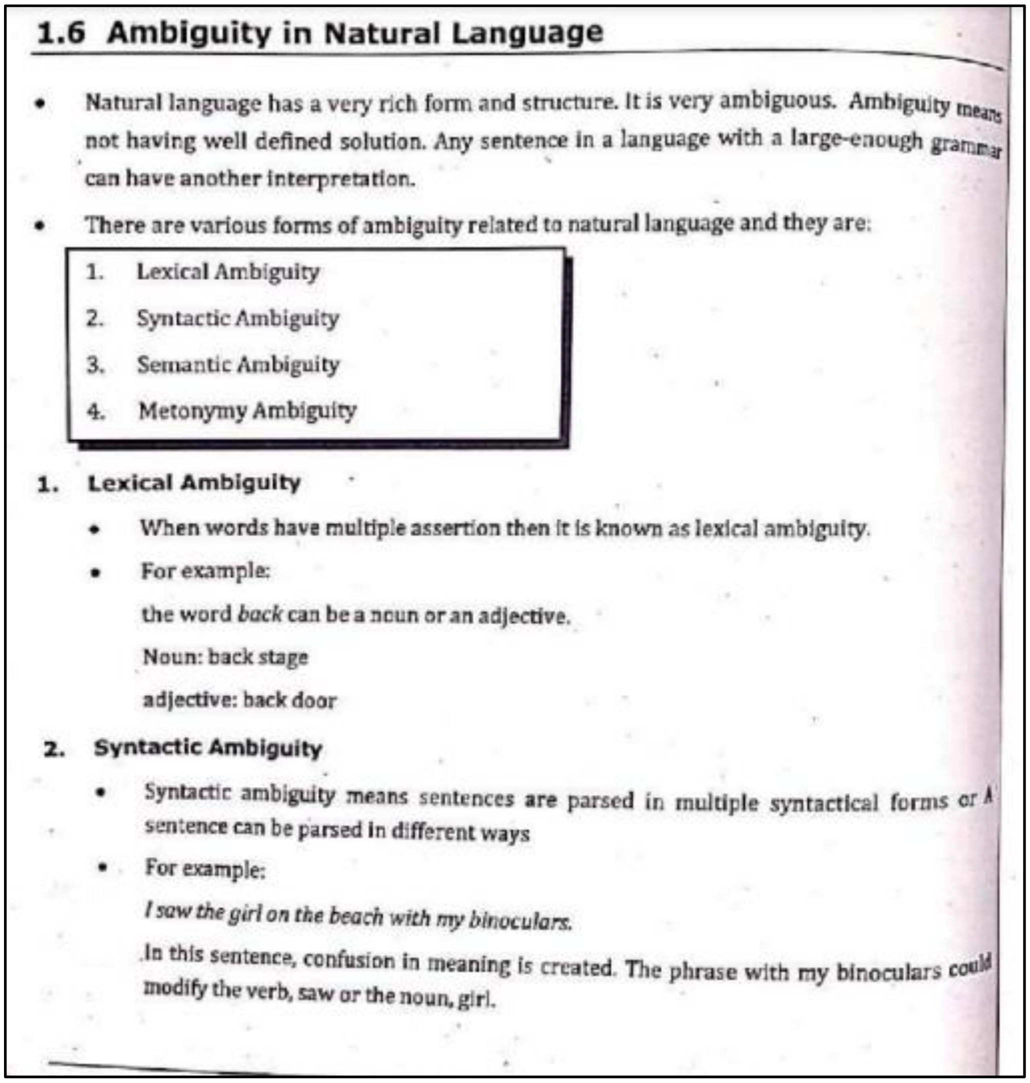
4. Discourse Integration

- The meaning of any sentence depends upon the meaning of the sentence just before it. Furthermore, it also brings about the meaning of immediately following sentence.
- For example: *Meena is a girl, she goes to school* here "she" is a dependency pointing to Meena.

5. Pragmatic Analysis

- During this, what was said is re-interpreted on what it truly meant. It contains deriving those aspects of language which necessitate real world knowledge.
- For example, *John saw Mary in a garden with a cat*, here we can't say that John is with cat or mary is with cat

2. Ambiguities in NLP



1.6 Ambiguity in Natural Language

- Natural language has a very rich form and structure. It is very ambiguous. Ambiguity means not having well defined solution. Any sentence in a language with a large-enough grammar can have another interpretation.
- There are various forms of ambiguity related to natural language and they are:

1. Lexical Ambiguity
2. Syntactic Ambiguity
3. Semantic Ambiguity
4. Metonymy Ambiguity

1. Lexical Ambiguity

- When words have multiple assertion then it is known as lexical ambiguity.
- For example:

  the word *back* can be a noun or an adjective.

  Noun: back stage

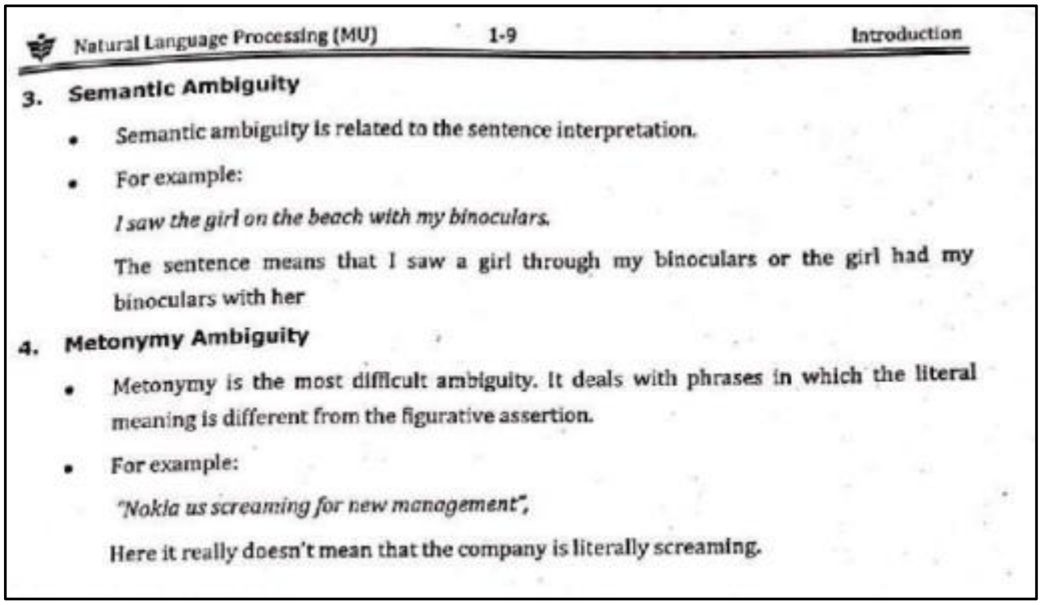
  adjective: back door

2. Syntactic Ambiguity

- Syntactic ambiguity means sentences are parsed in multiple syntactical forms or A sentence can be parsed in different ways
- For example:

  I saw the girl on the beach with my binoculars.

  In this sentence, confusion in meaning is created. The phrase with my binoculars could modify the verb, saw or the noun, girl.



3. Semantic Ambiguity

- Semantic ambiguity is related to the sentence interpretation.
- For example:

  I saw the girl on the beach with my binoculars.

  The sentence means that I saw a girl through my binoculars or the girl had my binoculars with her

4. Metonymy Ambiguity

- Metonymy is the most difficult ambiguity. It deals with phrases in which the literal meaning is different from the figurative assertion.
- For example:

  "Nokia us screaming for new management",

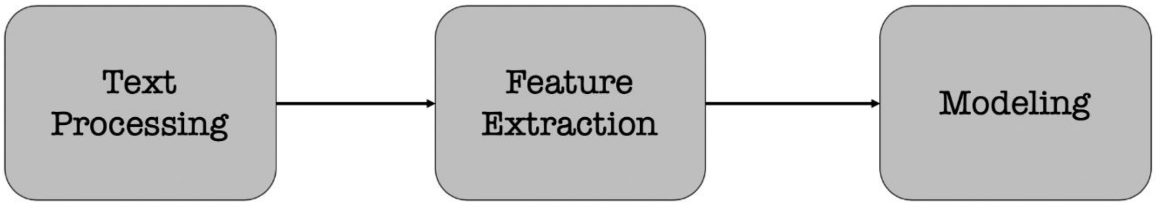
  Here it really doesn't mean that the company is literally screaming.

3. 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
2. Feature Extraction
   - Bag of Words d
   - TF-IDF
   - One-hot Encoding
   - Word Embeddings
3. Modeling



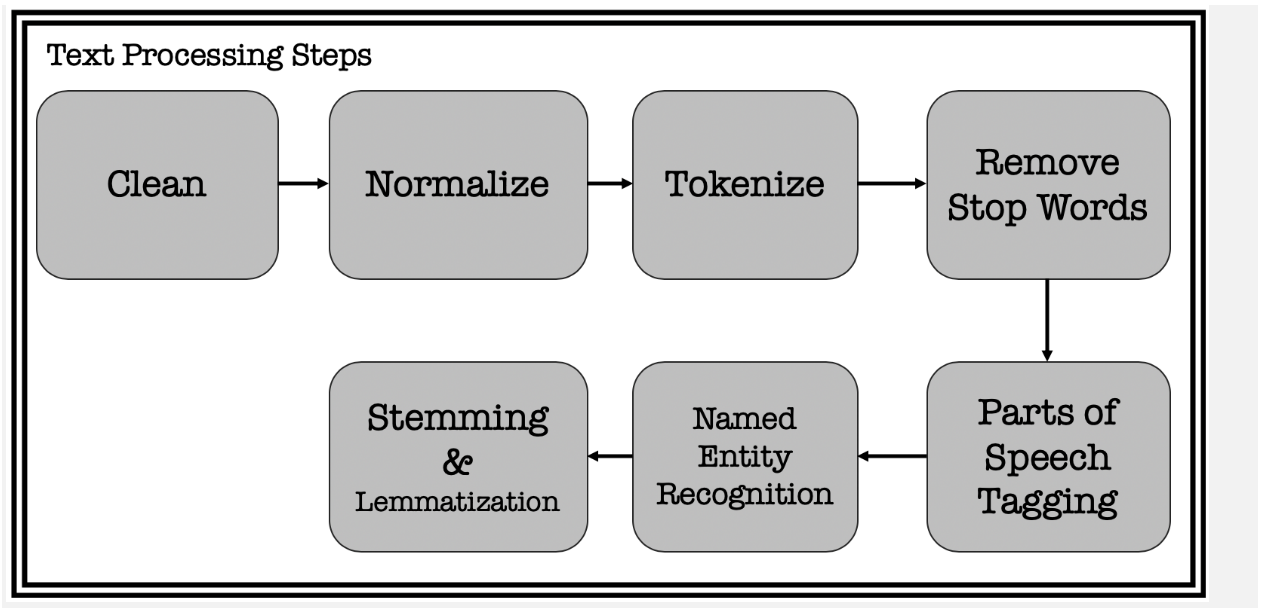
Each stage transforms text in some way and produces an intermediate result that the next stage needs. For example,

- **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 processing is 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, 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.

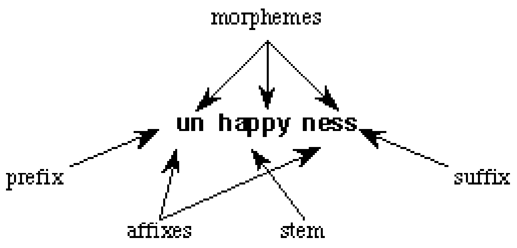
Modeling

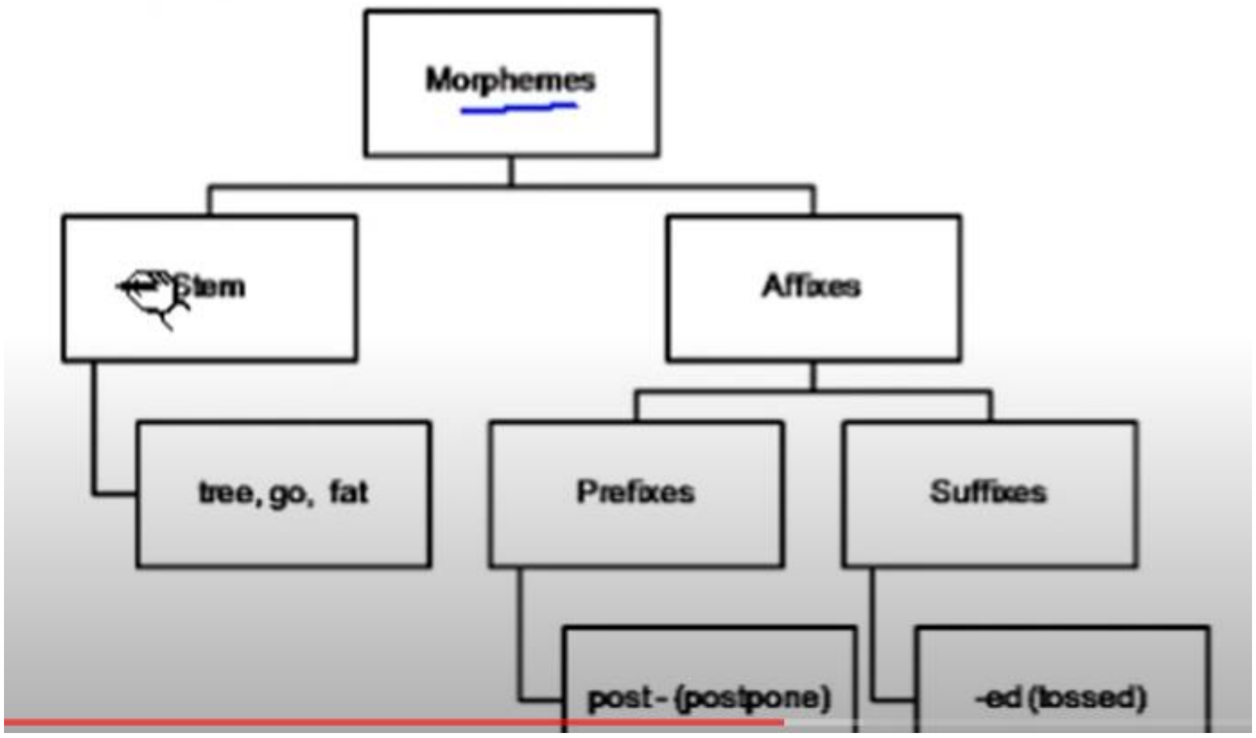
The final stage of the NLP pipeline is **modeling**, which includes designing a statistical or machine learning model, fitting its parameters to training data, using an optimization procedure, and then using it to make predictions about unseen data.

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** 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]

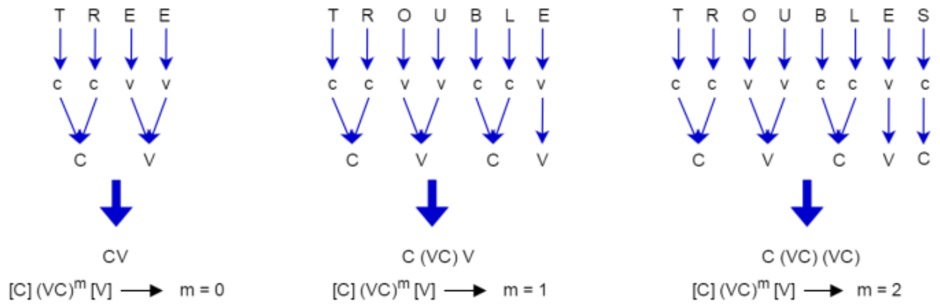
Here the square brackets denote arbitrary presence of consonants or vowels.
$(VC)^m$ denotes VC repeated m times. So the above expression can be written as,

$$[C](VC)^m[V]$$

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

5. SSES → SS
6. IES → I
7. SS → SS
8. S →

Step 1b

9. (m>0) EED → EE
10. (*v*) ED →
11. (*v*) ING →

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

12. AT → ATE
13. BL → BLE
14. IZ → IZE
15. (*d and not (*L or *S or *Z)) → single letter
16. (m=1 and *o) → E

Step 1c

17. (*v*) Y → I

Step 2

18. (m>0) ATIONAL → ATE
19. (m>0) TIONAL → TION
20. (m>0) ENCI → ENCE
21. (m>0) ANCI → ANCE
22. (m>0) IZER → IZE
23. (m>0) ABLI → ABLE
24. (m>0) ALLI → AL
25. (m>0) ENTLI → ENT

26.	(m>0) ELI	→	E
27.	(m>0) OUSLI	→	OUS
28.	(m>0) IZATION	→	IZE
29.	(m>0) ATION	→	ATE
30.	(m>0) ATOR	→	ATE
31.	(m>0) ALISM	→	AL
32.	(m>0) IVENESS	→	IVE
33.	(m>0) FULNESS	→	FUL
34.	(m>0) OUSNESS	→	OUS
35.	(m>0) ALITI	→	AL
36.	(m>0) IVITI	→	IVE
37.	(m>0) BILITI	→	BLE

Step 3

38.	(m>0) ICATE	→	IC
39.	(m>0) ATIVE	→	
40.	(m>0) ALIZE	→	AL
41.	(m>0) ICITI	→	IC
42.	(m>0) ICAL	→	IC
43.	(m>0) FUL	→	
44.	(m>0) NESS	→	

Step 4

45.	(m>1) AL	→	
46.	(m>1) ANCE	→	
47.	(m>1) ENCE	→	
48.	(m>1) ER	→	
49.	(m>1) IC	→	
50.	(m>1) ABLE	→	
51.	(m>1) IBLE	→	
52.	(m>1) ANT	→	
53.	(m>1) EMENT	→	
54.	(m>1) MENT	→	
55.	(m>1) ENT	→	
56.	(m>1 and (*S or *T)) ION	→	
57.	(m>1) OU	→	
58.	(m>1) ISM	→	
59.	(m>1) ATE	→	
60.	(m>1) ITI	→	
61.	(m>1) OUS	→	
62.	(m>1) IVE	→	

63. (m>1) IZE →

Step 5a

64. (m>1) E →
65. (m=1 and not *o) E →

Step 5b

66. (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).

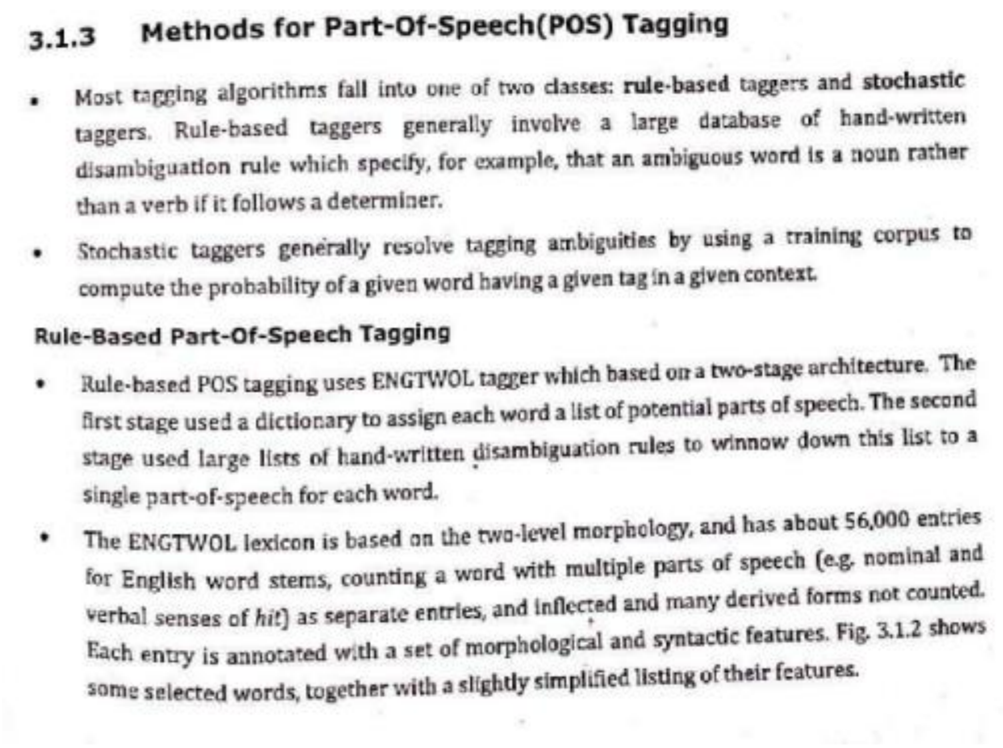
- No change will happen to the stem in other steps.
- Finally the output will be **CHARACTER**.

CHARACTERIZATION → CHARACTERIZE → **CHARACTER**

N-gram Language models:
3.pdf (stanford.edu) (text book)
Ngrams.2013.ppt (syr.edu)(ppt)

Module 3

1. Types of POS Taggers



3.1.3 Methods for Part-Of-Speech(POS) Tagging

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

Rule-Based Part-Of-Speech Tagging

- Rule-based POS tagging uses ENGTWOL tagger which based on a two-stage architecture. The first stage used a dictionary to assign each word a list of potential parts of speech. The second stage used large lists of hand-written disambiguation rules to winnow down this list to a single part-of-speech for each word.
- The ENGTWOL lexicon is based on the two-level morphology, and has about 56,000 entries for English word stems, counting a word with multiple parts of speech (e.g. nominal and verbal senses of *hit*) as separate entries, and inflected and many derived forms not counted. Each entry is annotated with a set of morphological and syntactic features. Fig. 3.1.2 shows some selected words, together with a slightly simplified listing of their features.

Word	POS	Additional POS features
Smaller	ADJ	COMPARATIVE
Entire	ADJ	ABSOLUTE ATTRIBUTIVE
Fast	ADV	SUPERLATIVE
That	DET	CENTRAL DEMONSTRATIVE SG
All	DET	PREDETERMINER SG/PL QUANTIFIER
Dog's	N	GENITIVE SG
Furniture	N	NOMINATIVE SG NONINDEFETERMINER
One-third	NUM	SG
She	PRON	PERSONAL FEMININE NOMINATIVE SG3
Show	V	IMPERATIVE VFIN
Show	V	PRESENT -SG3 VFIN
Shown	PCP2	SVOO SVO SV
Occurred	PCP2	SV
Occurred	V	PAST VFIN SV

Fig. 3.3 Sample lexical entries from the ENGTWOL lexicon.

1. In the first stage of the tagger, each word is run through the two-level lexicon transducer and the entries for all possible parts of speech are returned.

Stage 1 of ENGTWOL Tagging

First Stage : Run words through FST morphological analyzer to get all parts of speech.

- Example sentence : Pavlov had shown that salivation ...

PAVLOV	**N SG PROPER**
HAVE	**V PAST VFIN SVO (verb with subject and object)**
HAVE	**PCP2(past participle) SVO**
SHOWN	**SHOW PCP2 SVO SV SVOO (verb with subject and two complements)**
THAT	**ADV**
	PRON DEM SG
	DET CENTRAL DEM SG
	CS (subordinating conjunction)
SALIVATION	**N SG**

- A set of about 1,100 constraints are then applied to the input sentence to rule out incorrect parts of speech; the boldfaced entries in the given above show the desired result, in which the preterite (not participle) tag is applied to *had*, and the complementizer (CS) tag is applied the *that*. The constraints are used in a negative way, to eliminate tags that are inconsistent with the context. In stage 2 of ENGTWOL tagging we can write rule for word "that" to eliminate unwanted tags which initially assigned to it in stage one.

Stage 2 of ENGTWOL Tagging

Second Stage : Apply NEGATIVE constraints.

- Example :

```
ADVERBIAL-THAT RULE
Given input: "that"
if
(+1 A/ADV/QUANT);  /*If next word is adj/adv/quantifier */
(+2 SENT-LIM)      ; /*following which is E-O-S */
(NOT -1 SVOC/A)    ; /*and the previous word is not averb like "consider" */
                   /* which allows adjective complements in "I consider that odd"*/
then eliminate non-ADV tags
else eliminate ADV tag
```

- The first two clauses of this rule check to see that the *that* directly precedes a sentence-final adjective, adverb, or quantifier. In all other cases the adverb reading is eliminated.

Stochastic Part-Of-Speech Tagging

Stochastic taggers generally resolve tagging ambiguities by using a training corpus to compute the probability of a given word having a given tag in a given context. Stochastic tagger called also **HMM tagger or a Maximum Likelihood Tagger, or a Markov model** HMM TAGGER **tagger**, based on the Hidden Markov Model.

The simplest stochastic tagger applies the following approaches for POS tagging –

1. Approach 1 : Word Frequency Approach

In this approach, the stochastic taggers disambiguate the words based on the probability that a word occurs with a particular tag. We can also say that the tag encountered most frequently with the word in the training set is the one assigned to an ambiguous instance of that word. The main issue with this approach is that it may yield inadmissible sequence of tags.

- Assign each word its most likely POS tag
    - o If w has tags t1, ..., tk, then can use

$P(ti \mid w) = c(w,ti)/(c(w,t1) + \ldots + c(w,tk))$, where

$c(w,ti)$ = number of times w/ti appears in the corpus

- o Success: 91% for English

- Example heat :: noun/89, verb/5

2. Apprach 2 : Tag Sequence Probabilities

It is another approach of stochastic tagging, where the tagger calculates the probability of a given sequence of tags occurring. It is also called n-gram approach. It is called so because the best tag for a given word is determined by the probability at which it occurs with the n previous tags.

1. Given : sequence of words W

$$W = w_1, w_2, \ldots, w_n \text{ (a sentence)}$$

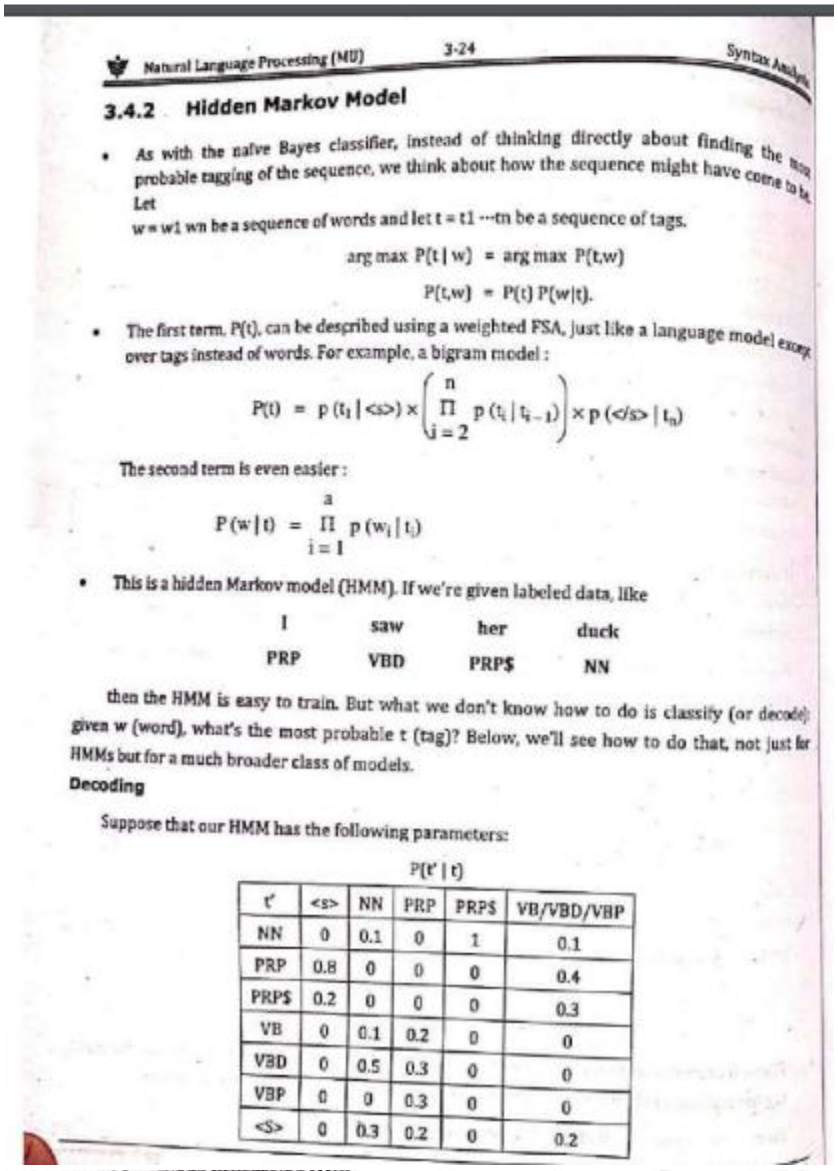
– e.g., W = heat water in a large vessel

2. Assign sequence of tags T :

$$T = t_1, t_2, \ldots, t_n$$

3. Find T that maximizes $P(T \mid W)$

2. How do you use HMM in POS Tagging



3.4.2 Hidden Markov Model

- As with the naïve Bayes classifier, instead of thinking directly about finding the most probable tagging of the sequence, we think about how the sequence might have come to be. Let
  w = w1 wn be a sequence of words and let t = t1 ---tn be a sequence of tags.

$$\arg\max P(t \mid w) = \arg\max P(t,w)$$

$$P(t,w) = P(t)\, P(w \mid t).$$

- The first term, P(t), can be desribed using a weighted FSA, just like a language model except over tags instead of words. For example, a bigram model :

$$P(t) = p(t_1 \mid \langle s \rangle) \times \left(\prod_{i=2}^{n} p(t_i \mid t_{i-1}) \right) \times p(\langle /s \rangle \mid t_n)$$

The second term is even easier :

$$P(w \mid t) = \prod_{i=1}^{a} p(w_i \mid t_i)$$

- This is a hidden Markov model (HMM). If we're given labeled data, like

I	saw	her	duck
PRP	VBD	PRP$	NN

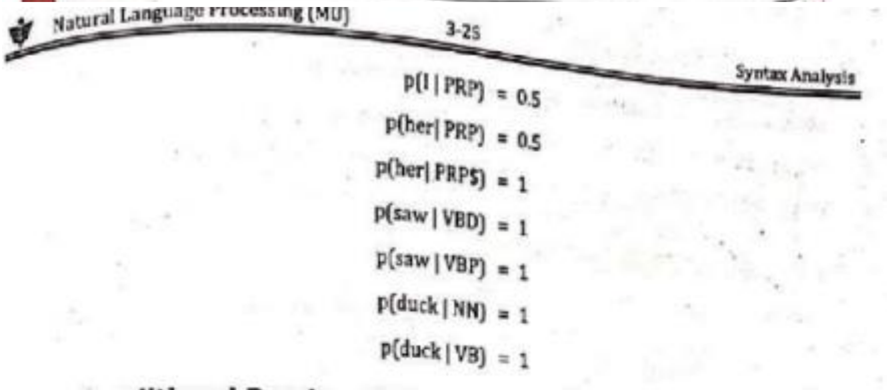
then the HMM is easy to train. But what we don't know how to do is classify (or decode): given w (word), what's the most probable t (tag)? Below, we'll see how to do that, not just for HMMs but for a much broader class of models.

Decoding

Suppose that our HMM has the following parameters:

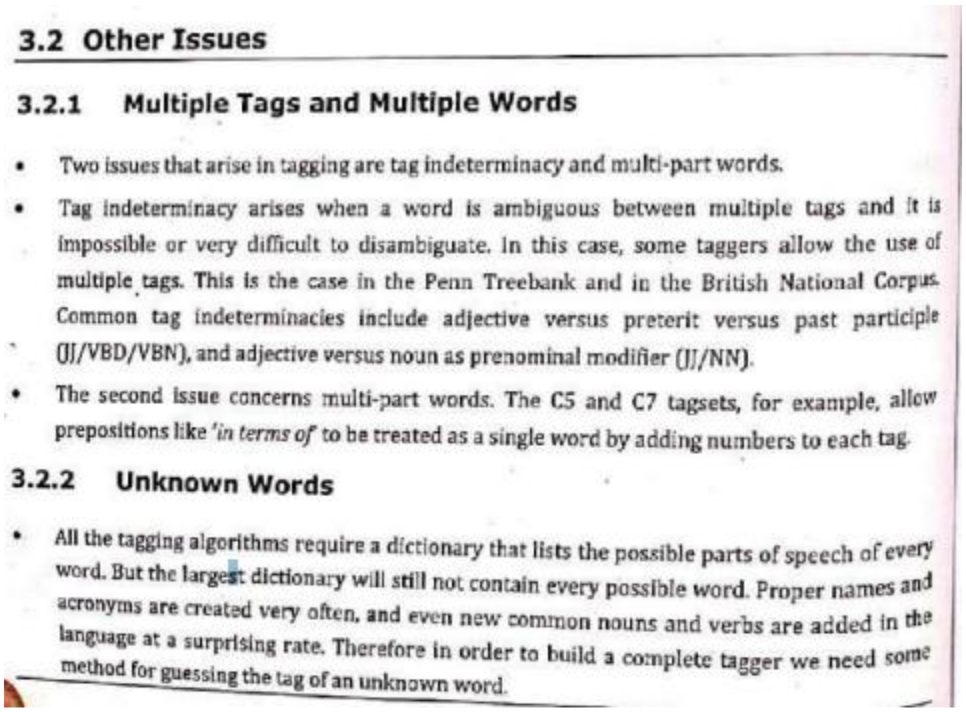
P(t' | t)

t'	<s>	NN	PRP	PRP$	VB/VBD/VBP
NN	0	0.1	0	1	0.1
PRP	0.8	0	0	0	0.4
PRP$	0.2	0	0	0	0.3
VB	0	0.1	0.2	0	0
VBD	0	0.5	0.3	0	0
VBP	0	0	0.3	0	0
<S>	0	0.3	0.2	0	0.2



p(I | PRP)	= 0.5
p(her| PRP)	= 0.5
p(her| PRP$)	= 1
p(saw | VBD)	= 1
p(saw | VBP)	= 1
p(duck | NN)	= 1
p(duck | VB)	= 1

3. Issues with HMM



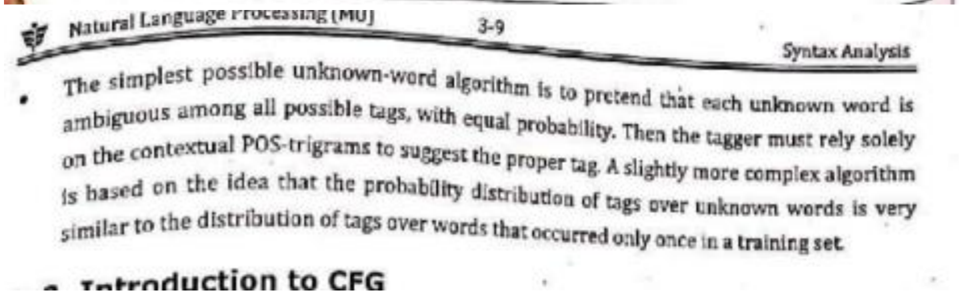
3.2 Other Issues

3.2.1 Multiple Tags and Multiple Words

- Two issues that arise in tagging are tag indeterminacy and multi-part words.
- Tag indeterminacy arises when a word is ambiguous between multiple tags and it is impossible or very difficult to disambiguate. In this case, some taggers allow the use of multiple tags. This is the case in the Penn Treebank and in the British National Corpus. Common tag indeterminacies include adjective versus preterit versus past participle (JJ/VBD/VBN), and adjective versus noun as prenominal modifier (JJ/NN).
- The second issue concerns multi-part words. The C5 and C7 tagsets, for example, allow prepositions like *'in terms of'* to be treated as a single word by adding numbers to each tag.

3.2.2 Unknown Words

- All the tagging algorithms require a dictionary that lists the possible parts of speech of every word. But the largest dictionary will still not contain every possible word. Proper names and acronyms are created very often, and even new common nouns and verbs are added in the language at a surprising rate. Therefore in order to build a complete tagger we need some method for guessing the tag of an unknown word.



- The simplest possible unknown-word algorithm is to pretend that each unknown word is ambiguous among all possible tags, with equal probability. Then the tagger must rely solely on the contextual POS-trigrams to suggest the proper tag. A slightly more complex algorithm is based on the idea that the probability distribution of tags over unknown words is very similar to the distribution of tags over words that occurred only once in a training set.

- Introduction to CFG

4. Assignment - 1

a. Differentiate b/w Interpolation and Backoffx
b. Viterbi algorithm is a variation of the forward algorithm which considers all words simultaneously in order to compute the most likely path.
c. 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>
d. Corpus: John read Moby Dick
   Mary read a different book
   She read a book by Cher

Test Data:

i. John read a book
ii. Cher read a book