

# M02S01-Introduction to Syntactic Processing

Natural Language Processing – Syntactic Processing

1. Introduction
2. The What and Why of Syntactic Processing
3. Parsing
4. Parts-of-Speech
5. Different Approaches to POS Tagging
6. Lexicon and Rule-based POS Tagging
7. Stochastic Parsing
8. The Viterbi Heuristic
9. Markov Chain and HMM
10. Explanation Problem
11. Learning HMM Model Parameters
12. HMM and the Viterbi Algorithm: Pseudocode
13. HMM & the Viterbi Algorithm: Python Implementation
14. Deep Learning Based POS Taggers
15. Summary
16. Graded Questions

## Introduction

Module on **Lexical Processing** focussed on text pre-processing and feature extraction techniques:

- Regular expressions
- Tokenization, Stemming, Lemmatization
- TF-IDF model
- Phonetic hashing
- The minimum edit distance algorithm

Built a spam detector and spell corrector in the module.

**In this module**, algorithms and techniques used to analyse the syntax or the grammatical structure of sentences.

1. First session - Basics of grammar (part-of-speech tags etc.) and writing own algorithms such as **HMMs to build POS taggers**.
2. Second session - Algorithms to **parse the grammatical structure** of sentences such as CFGs, PCFGs and dependency parsing.
3. Third session - Build an **Information Extraction (IE)** system to parse flight booking queries for users using techniques such as **Named Entity Recognition (NER)**.

Also, a class of models called **Conditional Random Fields (CRFs)**, widely used for building NER systems.

All these techniques fall under **syntactic processing**.

Syntactic processing is widely used in applications such as question answering systems, information extraction, sentiment analysis, grammar checking etc.

## In this session

This session will introduce:

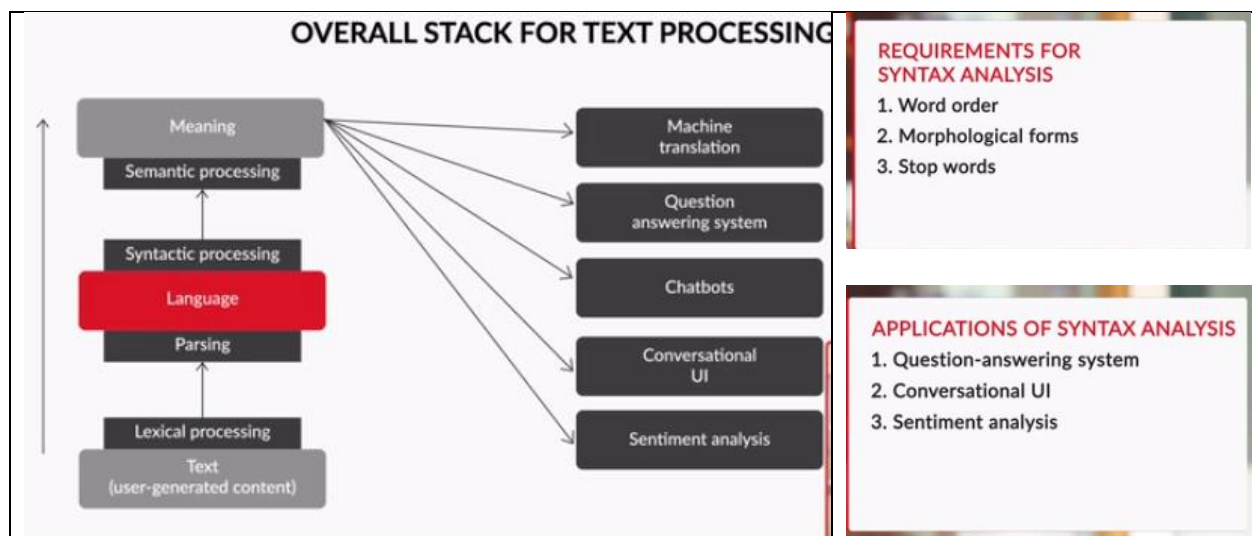
- The What and Why of Syntactic Processing
- Basics of Grammar and Parsing
- Algorithms for Part of Speech Tagging and Hidden Markov Models

## The What and Why of Syntactic Processing

Let's start with an example to understand **Syntactic Processing**:

- Canberra is the capital of Australia.
- Is Canberra the of Australia capital.

Both sentences have the same set of words, but only the first one is syntactically correct and comprehensible. Basic lexical processing techniques wouldn't be able to tell this difference. Therefore, more sophisticated syntactic processing techniques are required to understand the relationship between individual words in the sentence.



<p><b>Syntactical Processing</b></p> <p>Consider the following two statements:</p> <p>I - Syntactic analysis is specified by tokenised words for data cleaning and feature extraction</p> <p>II - Lexical analysis is specified by parse trees for checking structural dependencies in a sentence.</p> <p>Which of the above statement (s) is/are correct?</p> <p><input type="radio"/> Only I</p> <p><input type="radio"/> Only II</p> <p><input type="radio"/> Both I and II</p> <p><input checked="" type="radio"/> Neither I nor II</p> <p><b>Feedback:</b></p> <p>Lexical analysis is data pre-processing and feature extraction step. It involves the analysis at word level. Syntactical analysis aims at finding structural relationships among the words of a sentence.</p>	<p><b>Syntactical Analysis</b></p> <p>Canonicalisation is used in syntactical processing to get the words into its base form. Select whether the statement is True/False.</p> <p><input type="radio"/> True</p> <p><input checked="" type="radio"/> False</p> <p><b>Feedback:</b></p> <p>Canonicalisation is the process of getting the word to its base form. Changing words would change the linguistic structure of the sentence.</p>
--	--

	<p><b>Syntactical Analysis</b></p> <p>Can you jot down some applications of syntactical analysis apart from the ones mentioned in the video?</p> <div style="border: 1px solid #ccc; padding: 5px; margin: 5px 0;"> <p>Chatbots, Automated mailer responses, Legal derivations, Book scanning</p> </div> <div style="display: flex; justify-content: space-between; font-size: 0.8em;"> <span>Words 8</span> <span>Note: Once submitted, answer is not editable.</span> </div> <div style="background-color: #fff9c4; padding: 10px; margin-top: 10px;"> <p> <b>Suggested Answer</b></p> <p>Grammar checking applications</p> <p>Identifying correct part-of-speech tag of a word (i.e. deciding whether 'cut' is a noun/verb in a sentence)</p> </div>
--	---

Lexical analysis aims at data cleaning and feature extraction, which it does by using techniques such as lemmatization, removing stopwords, rectifying misspelt words, etc. In **syntactic analysis**, our aim will be to understand the roles played by the words in the sentence, the relationship between words and to parse the grammatical structure of sentences.

Next is which lexical processing techniques are irrelevant for syntactic processing.

## WHY SYNTAX ANALYSIS?

UpGrad

- Word orders and meaning
  - Dog bites man
  - Man bites dog
- Role of "stop words"
  - Tendulkar lost to Australia
  - Tendulkar lost in Australia

- Role of morphological forms
  - Our workers are working hard to make our code work
- Role of parts of speech
  - My uncle is learning driving in a driving school
- Dependencies
  - What is the capital of India?
  - What is the name of the country in whose capital India led the UN delegation on climate change?

## Syntactical Analysis

Syntactical analysis looks at the following aspects in the sentence which lexical doesn't. (Multiple options are correct)

☒ Words order and meaning

✓ Correct

💡 Feedback :

Syntactical analysis aims to find how words are dependent on each other. Changing word order will make it difficult to comprehend the sentence

☒ Retaining stopwords

✓ Correct

💡 Feedback :

Removing stopwords can altogether change the meaning of a sentence.

☒ Morphology of words

✓ Correct

💡 Feedback :

Stemming, lemmatisation will bring the words to its base form, thus modifying the grammar of the sentence.

☒ Parts-of-speech of words in a sentence

✓ Correct

💡 Feedback :

Identifying correct part-of-speech of a word is important. Example:

'cuts and bruises on his face' (Here 'cuts' is a noun)

'he cuts an apple' (Here, 'cuts' is a verb)

After basic idea of syntactic processing, let's study the different levels of syntactic analysis.

Download and install [the NLTK toolkit](#) on your system, if not already done.

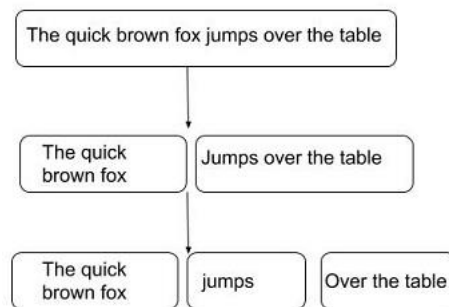
## Parsing

A key task in syntactical processing is **parsing**. It means to break down a given sentence into its 'grammatical constituents'. Parsing is an important step in many applications which helps us better understand the linguistic structure of sentences.

Let's ask a question answering (QA) system, Alexa or Siri: "Who won the cricket world cup in 2015?"

The QA system can respond meaningfully only if it understands that 'cricket world cup' is related to 'in 2015'. 'in 2015' refers to a specific time frame, and thus modifies the question significantly. Finding such dependencies or relations between the phrases of a sentence can be achieved using parsing techniques.

A parsed sentence: "The quick brown fox jumps over the table". Below show the three main **constituents** of this sentence. Actual parse trees are different from the simplified representation below.



### Parsing

This structure divides the sentence into three main constituents:

- 'The quick brown fox' is a **noun phrase**
- 'jumps' is a **verb phrase**
- 'over the table' is a **prepositional phrase**.

More on elements of grammar and parsing techniques in the segments that follow. Following different **levels of syntactical analysis**:

- Part-of-speech tagging
- Constituency parsing
- Dependency parsing

<b>LEVELS OF SYNTAX ANALYSIS</b> 1. Part-of-speech (POS) tagging a. Tagging as verb/noun/adjective, etc. b. Based on the linguistic role of the word c. Also called 'shallow parsing'	<b>LEVELS OF SYNTAX ANALYSIS</b> 1. Part-of-speech (POS) tagging 2. Constituency/Paradigmatic relationship a. Based on the linguistic role of the subset of words b. Context-free grammars	<b>LEVELS OF SYNTAX ANALYSIS</b> 1. Part-of-speech (POS) tagging 2. Constituency/Paradigmatic relationship 3. Dependencies a. Subject, object, modifier, etc <ul style="list-style-type: none"> <li>The animals that were trapped in the rapidly spreading forest fire, and that spent three days without food or shelter on an isolated barn, were finally <b>rescued yesterday</b></li> </ul>
---	--	--

To summarise, following levels of syntactic analysis:

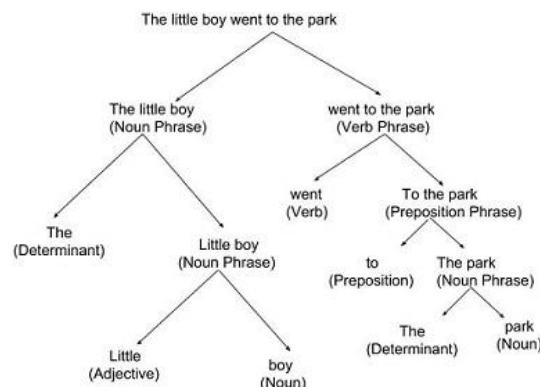
- Part-of-speech (POS) tagging
- Constituency parsing
- Dependency parsing

Using an example sentence: "The little boy went to the park."

**POS tagging** is the task of assigning a part of speech tag (POS tag) to each word. The POS tags identify the linguistic role of the word in the sentence. The POS tags of the sentence are:

The	little	boy	went	to	the	park
<b>Determinant</b>	<b>Adjective</b>	<b>Noun</b>	<b>Verb</b>	<b>Preposition</b>	<b>Determinant</b>	<b>Noun</b>

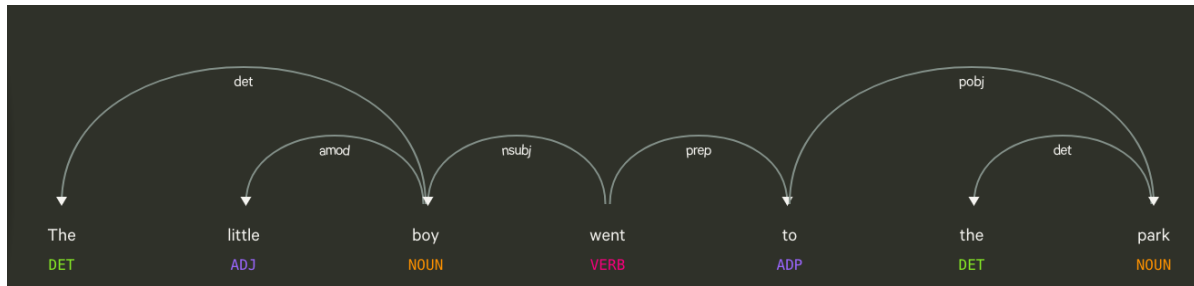
**Constituency parsers** divide the sentence into constituent phrases such as noun phrase, verb phrase, prepositional phrase etc. Each constituent phrase can itself be divided into further phrases. The constituency parse tree given below divides the sentence into two main phrases - a noun phrase and a verb phrase. The verb phrase is further divided into a verb and a prepositional phrase, and so on.



### Constituency Parsing

**Dependency Parsers** do not divide a sentence into constituent phrases, but rather establish relationships directly between the words themselves. The figure below is an example of a dependency parse tree of the sentence given above (generated using the [spaCy dependency](#)

[visualiser](#)). In this module, you'll understand when dependency parsing is more useful than constituency parsing and study the elements of dependency grammar.



## Dependency Parse

You will study these parsing techniques in the sections that follow. In the next few segments, you will study POS tagging in detail.

<p><b>Parsing</b></p> <p>Which of the following is not a technique of Syntactic Processing:(Multiple options are correct)</p> <p><input type="checkbox"/> POS tagging</p> <p><input checked="" type="checkbox"/> Stemming</p> <p><b>Feedback :</b></p> <p>Stemming changes the morphology of words, thereby changing its linguistic role in a sentence</p> <p><input checked="" type="checkbox"/> Stopwords removal</p> <p><b>Feedback :</b></p> <p>Stopwords removal makes the sentence syntactically incorrect</p> <p><input type="checkbox"/> Dependency Parsing</p>	<p><b>Parsing</b></p> <p>Constituency parsing is known as Paradigmatic parsing</p> <p><input type="radio"/> True</p> <p><b>Feedback :</b></p> <p>Constituency parsing identifies constituents which are in the same paradigm</p> <p><input checked="" type="radio"/> False</p> <p><b>Feedback :</b></p> <p>Constituency parsing identifies constituents which are in the same paradigm</p>
---	--



## Parts-of-Speech

First level of syntactic analysis- **POS (parts-of-speech) tagging**. A word can be tagged as a noun, verb, adjective, adverb, preposition etc. depending upon its role in the sentence. Assigning the correct tag such as noun, verb, adjective etc. is one of the most fundamental tasks in syntactic analysis.

Ask smart home device a question - "Ok Google, where can I get the permit to work in Australia?". Now, the word 'permit' can potentially have two POS tags - noun and a verb. In the phrase 'I need a work permit', the correct tag of 'permit' is 'noun'. On the other hand, in the phrase "Please permit me to take the exam.", the word 'permit' is a 'verb'.

Assigning the correct POS tags helps us better understand the intended meaning of a phrase or a sentence and is thus a crucial part of syntactic processing. In fact, all the subsequent parsing techniques (constituency parsing, dependency parsing etc.) use the part-of-speech tags to parse the sentence.

Next is commonly occurring parts of speech tags.

<b>PART-OF-SPEECH TAGGING</b> 1. Also called 'shallow parsing' 2. What is a 'part of speech'? a. Syntactic class of a term b. Based on role played in sentence c. Nouns, verbs, adjectives, adverbs, prepositions, etc.	<b>PART-OF-SPEECH TAGGING</b> 1. Also called 'shallow parsing' 2. What is a 'part of speech'? 3. Terms of the same parts of speech can be substituted with one another to make a syntactically correct sentence a. Example: My brother goes to school	<b>PART-OF-SPEECH TAGGING</b> 1. Also called 'shallow parsing' 2. What is a 'part of speech'? 3. Terms of the same parts of speech can be substituted with one another to make a syntactically correct sentence a. Example: My cat goes to school
--	---	---

PART-OF-SPEECH TAGS				No need to remember all the POS tags except for a few. Most will be known when working on the problems, but it's important to be aware of all the types of tags.
Part of Speech	Tag	Definition	Example	
Noun	NN	Represent entities and concepts	People (Ronnie) Animals (cat) Things (table) Etc.	
Proper noun	NNP	Names of entities	City Names (Chennai, Mumbai, Delhi)	
Adverbial noun	NR	Nouns that modify other parts of speech	North Delhi, yesterday	
Pronoun	PP	Nouns that are salient in a discourse context	He, she He went to school. <u>His</u> name is <u>Ram</u> .	
Subject Case Pronoun	PPS	Pronouns that form the subject	They went to school	
Object Case Pronoun	PPO	Pronouns that form the object	The bus went to them	
Possessive Pronoun	PP\$	Object describes a possessor	My car	
Reflexive Pronoun	PPL	Refers to a possession relationship	Myself, himself, herself	

## PART-OF-SPEECH TAGS

Part of Speech	Tag	Definition	Example
Determiner	DT	Describes a particular reference to a noun	This, That
Article	AT	Special case of determiners	A, an, the
Adjective	JJ	Describes properties of a noun	Red rose Fast car Lazy cat
Interrogative pronoun	WDT	Wh- determiner Word that starts a question	Which, what, whom Which city did you come from? What did you do?
Verb	VB	Word that describes an action Can also represent assertion	Ran, threw, walked The capital of India is Delhi
Adverb	RB	Word that modifies a verb	I rarely travel abroad
Preposition	IN	Represents a spatial relationship	Under the bus Over the city In the car
Conjunction	CC	Connects two parts of a sentence	And, because You cannot start a sentence with because because because is a conjunction

## LIST OF ALL POS TAGS

UpGrad

### Penn Treebank Project

Number	Tag	Description	Number	Tag	Description
1.	CC	Coordinating conjunction	19.	PRP\$	Possessive pronoun
2.	CD	Cardinal number	20.	RB	Adverb
3.	DT	Determiner	21.	RBR	Adverb, comparative
4.	EX	Existential there	22.	RBS	Adverb, superlative
5.	FW	Foreign word	23.	RP	Particle
6.	IN	Preposition or subordinating conjunction	24.	SYM	Symbol
7.	JJ	Adjective	25.	TO	to
8.	JJR	Adjective, comparative	26.	UH	Interjection
9.	JJS	Adjective, superlative	27.	VB	Verb, base form
10.	LS	List item marker	28.	VBD	Verb, past tense
11.	MD	Modal	29.	VBG	Verb, gerund or present participle
12.	NN	Noun, singular, or mass	30.	VBN	Verb, past participle
13.	NNS	Noun, plural	31.	VBP	Verb, non-3rd person singular present
14.	NNP	Proper noun, singular	32.	VBZ	Verb, 3rd person singular present
15.	NNPS	Proper noun, plural	33.	WDT	Wh-determiner
16.	PDT	Predeterminer	34.	WP	Wh-pronoun
17.	POS	Possessive ending	35.	WP\$	Possessive wh-pronoun
18.	PRP	Personal pronoun	36.	WRB	Wh-adverb

There are 36 POS tags in the Penn treebank in NLTK. It is recommended to remember these common tags. It'll help avoid the trouble of looking up meanings of tags to focus on the core concepts.

Note that the set of POS tags is not standard - some books/applications may use only the base forms such as NN, VB, JJ etc without using granular forms, though NLTK uses [this set of tags](#). Since we'll use NLTK heavily, we recommend reading this list of tags at least once.

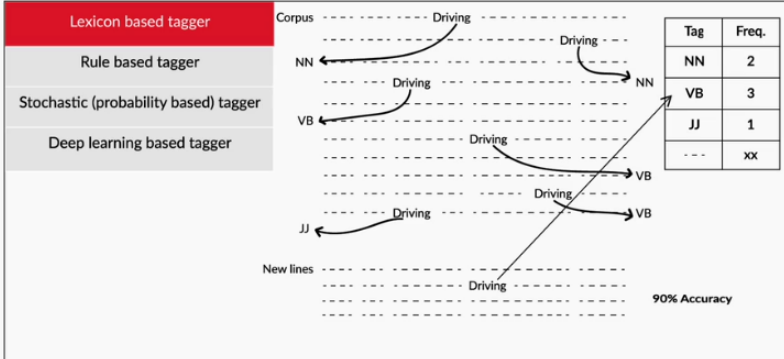
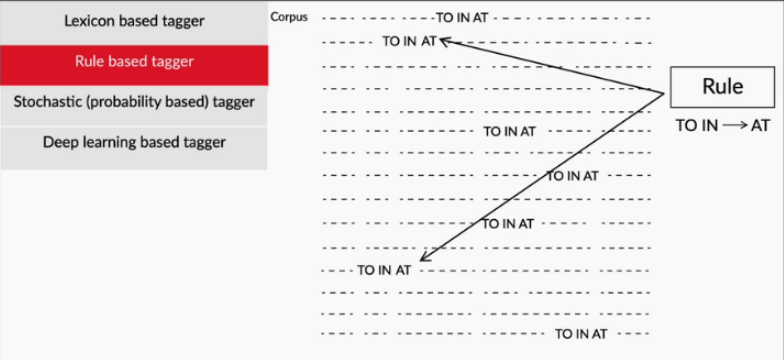
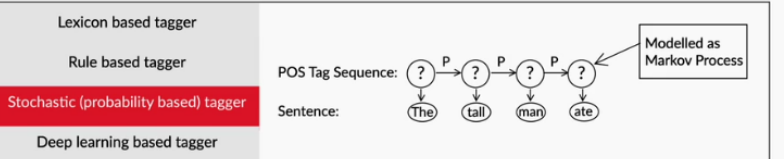
<p><b>POS Tagging</b></p> <p>Consider the following two sentences -</p> <p>My brother goes to school,</p> <p>My table goes to school.</p> <p>Select all the options that apply. (Multiple answers are correct)</p> <div> <input checked="" type="checkbox"/> Both brother and table are nouns.         </div> <div> <p><b>Feedback :</b></p> <p>Common noun: brother and table. One or more options are correct</p> </div> <div> <input checked="" type="checkbox"/> "My table goes to school" is a syntactically correct sentence         </div> <div> <p><b>Feedback :</b></p> <p>This is a correct grammatical sentence. One or more options are correct</p> </div> <div> <input type="checkbox"/> 'My table goes to school' is a syntactically incorrect sentence because it has no meaning         </div> <div> <input type="checkbox"/> All of the above         </div>	<p><b>POS Tagging</b></p> <p>Select the most appropriate tag sequence for the given sentence:</p> <p>People continue to inquire the reason for the race for outer space.</p> <div> <input checked="" type="radio"/> People/NNS continue/VBP to/TO inquire/VB the/DT reason/NN for/IN the/DT race/NN for/IN outer/JJ space/NN         </div> <div> <p><b>Feedback :</b></p> <p>VBP is used with non -3rd person and is in the present tense.</p> </div> <div> <input type="radio"/> People/NNS continue/VBD to/TO inquire/VB the/DT reason/NN for/IN the/DT race/NN for/CC outer/JJ space/NN         </div> <div> <input type="radio"/> People/NNS continue/VBD to/TO inquire/VB the/DT reason/NN for/IN the/DT race/NN for/IN outer/RB space/NN         </div> <div> <input type="radio"/> People/NNS continue/VBP to/TO inquire/VB the/DT reason/NN for/IN the/DT race/NN for/IN outer/RB space/NN         </div>
<p><b>Syntactical Analysis</b></p> <p>What is the 'must have' condition for a sentence to be syntactically correct?</p> <div> <input type="radio"/> The sentence should be meaningful         </div> <div> <input checked="" type="radio"/> The sentence should have the 'right grammar' - i.e. the nouns, adjectives, verbs are all placed correctly in the sentence         </div> <div> <p><b>Feedback :</b></p> <p>Syntactics check the grammatical structure of the sentence</p> </div> <div> <input type="radio"/> The sentence should be both meaningful and have the right syntax         </div> <div> <input type="radio"/> None of the above         </div>	<p><b>POS Tagging</b></p> <p>A and B are the POS tags for the word 'wound'. Which of the following option is correct?</p> <p>The bandage was wound/A around the wound/B.</p> <div> <input type="radio"/> A-Verb, B-Verb         </div> <div> <input type="radio"/> A-Noun, B-Verb         </div> <div> <input type="radio"/> A- Noun, B-Noun         </div> <div> <input checked="" type="radio"/> A- Verb, B-Noun         </div> <div> <p><b>Feedback :</b></p> <p>One of the 'wound' represents the injury/cut. Other represents the act of 'winding' the bandage</p> </div>
<p><b>POS Tagging</b></p> <p>Run the following commands in Jupyter notebook:</p> <pre>import nltk nltk.help.pupenn_tagset()</pre> <p>Which of following are sentence terminators? (Guide for downloading and installing NLTK is shared in the segment 'The What and Why of Syntactic Analysis') (Multiple options are correct)</p> <div> <input checked="" type="checkbox"/> . <span>✓ Correct</span> </div> <div> <p><b>Feedback :</b></p> <p>Full stop terminates the sentence</p> </div> <div> <input type="checkbox"/> ,         </div> <div> <input checked="" type="checkbox"/> ! <span>✓ Correct</span> </div> <div> <p><b>Feedback :</b></p> <p>Exclamation mark terminates exclamatory sentences</p> </div> <div> <input checked="" type="checkbox"/> ? <span>✓ Correct</span> </div> <div> <p><b>Feedback :</b></p> <p>Question mark terminates interrogative sentences</p> </div>	

## Different Approaches to POS Tagging

Discuss techniques and algorithms for POS tagging. Following four main techniques used for POS tagging:

- Lexicon-based
- Rule-based
- Probabilistic (or stochastic) techniques
- Deep learning techniques

This session will cover the first three tagging approaches in detail and the basics of deep-learning based taggers. Deep-learning based models will be covered in detail in the Neural Networks course.

<p><b>DIFFERENT APPROACHES TO POS TAGGING</b></p> <p>UpGrad</p> <div> <p>Lexicon based tagger</p> <p>Rule based tagger</p> <p>Stochastic (probability based) tagger</p> <p>Deep learning based tagger</p> </div> 	<p>for each word, it assigns the POS tag that most frequently occurs for that word in some training corpus. Such a tagging approach cannot handle unknown/ambiguous words. For example:</p> <ul style="list-style-type: none"> <li>• I went for a <b>run/NN</b></li> <li>• I <b>run/VB</b> in the morning</li> </ul> <p>Lexicon tagger will tag 'run' basis the highest frequency tag. In most contexts, 'run' is likely to appear as a verb, implying that 'run' will be wrongly tagged in the first sentence.</p>
<p><b>DIFFERENT APPROACHES TO POS TAGGING</b></p> <p>UpGrad</p> <div> <p>Lexicon based tagger</p> <p>Rule based tagger</p> <p>Stochastic (probability based) tagger</p> <p>Deep learning based tagger</p> </div> 	<p>If there's a rule that is applied to the entire text, such as, 'replace VB with NN if the previous tag is DT', or 'tag all words ending with <i>ing</i> as VBG', the tag can be corrected. <b>Rule-based</b> tagging methods use such an approach.</p> <p>Rule-based taggers first use lexicon based tagging and then correct the incorrect tags (if any) basis some predefined rules. Rules are generally defined as “<i>Replace Tag1 with Tag2 in some context</i>”. The <i>context</i> could be the tag of the preceding word or word ending with (like ‘ous’, ‘es’).</p>
<p><b>DIFFERENT APPROACHES TO POS TAGGING</b></p> <p>UpGrad</p> <div> <p>Lexicon based tagger</p> <p>Rule based tagger</p> <p>Stochastic (probability based) tagger</p> <p>Deep learning based tagger</p> </div> 	<p><b>Probabilistic taggers</b> don't naively assign the highest frequency tag to each word, instead, they look at slightly longer parts of the sequence and often use the tag(s) and the word(s) appearing before the target word to be tagged.</p>

### POS Tagging approaches

Rule-based taggers first use lexicon based tagging and then correct the incorrect tags (if any) basis some predefined rules. Rules are generally defined as "Replace Tag1 with Tag2 in some context". The context could be the tag of the preceding word or word ending with (like 'ous', 'es'). Following is a simple rule-based tagger for the below sentence.

Sentence: University/**NN** promised/**VB** to/**TO** increase/**NN** grants/**NNS** to/**TO** professors/**NNS** with/**IN** fabulous/**NN** records/**NNS**

Rules:

1. Replace NN with VB when the previous word is TO
2. Replace tag TO with IN when the next tag is NNS
3. Replace the tags of word ending with 'ous' to JJ

Select the correct tag sequence for highlighted observation after applying the three rules:

- ☐ University/**NN** promised/**VB** to/**TO** increase/**NN** grants/**NNS** to/**TO** professors/**NNS** with/**IN** fabulous/**NN** records/**NNS**
- ☐ University/**NN** promised/**VB** to/**IN** increase/**NN** grants/**NNS** to/**TO** professors/**VB** with/**IN** fabulous/**JJ** records/**NNS**
- ☐ University/**NN** promised/**VB** to/**TO** increase/**VB** grants/**NNS** to/**TO** professors/**NNS** with/**IN** fabulous/**JJ** records/**NNS**

- ☒ University/**NN** promised/**VB** to/**TO** increase/**VB** grants/**NNS** to/**IN** professors/**NNS** with/**IN** fabulous/**JJ** records/**NNS** ✓ **Correct**

Feedback :

All the rules are applied

### POS Tagging approach

Given the POS tagged corpora:

"You/**PRP** always/**RB** chose/**VBD** the/**DT** wrong/**JJ** road/**NN**.

Rohit/**NN** always/**RB** used/**VBD** the/**DT** wrong/**JJ** technique/**NN**.

You/**PRP** wrong/**VB** me by your suspicions.

Edward often/**RB** spells/**VN** the words wrong/**RB**".

For the sentence: "Wrong seldom prospers". Lexicon based POS tag for 'wrong' is:

☐ Noun

☒ Adjective

Feedback :

The word 'wrong' is tagged as an adjective maximum number of times

☐ Verb

☐ Adverb

### POS Tagging approach

Rule-based taggers are rigid and likely to underperform compared to probabilistic taggers. True/ False

☒ True

Feedback :

One single rule can lower the entire accuracy.

For example: change the tags of words ending with 'ing' to 'VBG'. But, the words like amazing, interesting, etc will also get tagged as VBD

☐ False

### POS Tagging approaches

Rule-based taggers are also known as syntagmatic taggers. Which of the following correctly explains the role of these taggers?

- ☒ Syntagmatic taggers first use statistical techniques to extract information from the training corpus and then use predefined rules to reduce mistakes committed by the statistical technique ✓ **Correct**

Feedback :

Rule-based taggers/syntagmatic taggers first tag the word using the lexical-based tagging approach and then applies the rules mentioned for changing the tag

☐ Syntagmatic tagger use only probabilistic method to tag the sequence of words

☐ Syntagmatic tagger use recurrent neural networks to tag the words in a sentence

## Lexicon and Rule-based POS Tagging

**Lexicon tagger** uses a simple statistical tagging algorithm: for each token, it assigns the most frequently assigned POS tag. For example, it will assign the tag "verb" to any occurrence of the word "run" if "run" is used as a verb more often than any other tag.

**Rule-based taggers** first assign the tag using the lexicon method and then apply predefined rules. Some examples of rules are:

1. Change the tag to VBG for words ending with ‘-ing’
2. Changes the tag to VBD for words ending with ‘-ed’
3. Replace VBD with VBN if the previous word is 'has/have/had'

Defining such rules require some exploratory data analysis and intuition.

Learn to implement the lexicon and rule-based tagger on the [Treebank corpus of NLTK](#). Let's first explore the corpus.

### Programming Exercise - Exploratory Analysis for POS Tagging

Practice exercise will use the Jupyter notebook to answer the questions. The notebook contains some starter code to read the data and explain its structure.

<p><b>Lexical &amp; Rule-based Tagging</b></p> <p>What percent of words with the tag 'VBD' (verb, past tense) end with 'ed'?</p> <p><input type="radio"/> 90%</p> <p><input type="radio"/> 20%</p> <p><input checked="" type="radio"/> 38%</p> <p><b>Feedback :</b></p> <p>Not all the words ending with 'ed' are Verbs. Making this as a rule for Rule based tagger could bring down the accuracy</p> <p><input type="radio"/> 50%</p>	<p><b>Lexical and Rule-based Tagging</b></p> <p>What percent words with the tag 'VBG' (verb, past tense) end with 'ing'?</p> <p><input type="radio"/> 10%</p> <p><input type="radio"/> 45%</p> <p><input type="radio"/> 36%</p> <p><input checked="" type="radio"/> 99%</p> <p><b>Feedback :</b></p> <p>This is quite high. We could use this as a rule to correct the tags of words if the words are ending with 'ing'</p>
<p><b>Lexical &amp; Rule-based Tagging</b></p> <p>What fraction of modals MD are followed by a verb VB? (MD are modal verbs like: will, shall, can)</p> <p><input checked="" type="radio"/> 82%</p> <p><b>Feedback :</b></p> <p>We could use this as a rule again. Try looking for words tagged as VBG, VBD, VBP. Replace them with VB if the previous tag is MD</p> <p><input type="radio"/> 20%</p> <p><input type="radio"/> 50%</p>	

## Stochastic Parsing

A commonly used probabilistic algorithm for POS tagging - the **Hidden Markov Model (HMM)**.

**Bayes' theorem** and the chain rule of probability. Two features  $X = (x_1, x_2)$  and a binary target variable  $y$  (class = 0/1). According to the **Bayes' rule**, the probability of a point  $(x_1, x_2)$  belonging to the class  $c_1$  is given by:

$$P(\text{class} = c_1 | x_1, x_2) = \frac{P(x_1, x_2 | c_1) \cdot P(c_1)}{P(x_1, x_2)}$$


According to the **chain rule of probability**,  $P(x_1, x_2 | c_1)$  is same as  $P(x_1 | c_1) \cdot P(x_2 | c_1)$ .

A similar idea is used by probabilistic parsers to assign POS tags to sequences of words. Say you want to tag the word sequence 'The high cost' and compute the probability that the tag sequence is (DT, JJ, NN). For simplicity, let's work with only these three POS tags, and also assume that only three tag sequences are possible - (DT, JJ, NN), (DT, NN, JJ) and (JJ, DT, NN).

The probability that the tag sequence is (DT, JJ, NN) can be computed as:

$$P(DT, JJ, NN | \text{The, high, cost}) = \frac{P(\text{The, high, cost} | DT, JJ, NN) \cdot P(DT, JJ, NN)}{P(\text{The, high, cost})}$$

Similarly, we can compute the probabilities of the other two sequences and assign the sequence that has the maximum probability. We will need to use some simplifying assumptions to compute the right-hand side of the equation. Let's see how.

<b>STOCHASTIC PARSING</b> Find the most likely parse sequence for "The high cost" "The"/POS tag? "high"/POS tag? "cost"/POS tag?  The/DT high/JJ cost/NN The/DT high/NN cost/JJ The/JJ high/DT cost/NN	$P(DT, JJ, NN   \text{"the", "high", "cost"})$ $= [P(\text{"the", "high", "cost"}   DT, JJ, NN) \cdot P(DT, JJ, NN)] / [P(\text{"the", "high", "cost"})]$ $= [P(\text{"the"}   DT, JJ, NN) \cdot P(\text{"high", "cost"}   \text{"the", DT, JJ, NN}) \cdot P(DT, JJ, NN)] / [P(\text{"the", "high", "cost"})]$
---	--



	$= [P(\text{"the"}   DT, JJ, NN) * P(\text{"high"}   \text{"the"}, DT, JJ, NN) * P(\text{"cost"}   \text{"the"}, \text{"high"}, DT, JJ, NN) * P(DT, JJ, NN)] / [P(\text{"the"}, \text{"high"}, \text{"cost"})]$ $= [P(\text{"the"}   DT) * P(\text{"high"}   JJ) * P(\text{"cost"}   NN) * P(DT, JJ, NN)] / [P(\text{"the"}, \text{"high"}, \text{"cost"})]$ $= [P(\text{"the"}   DT) * P(\text{"high"}   JJ) * P(\text{"cost"}   NN) * P(DT) * P(JJ   DT) * P(NN   DT, JJ)] / [P(\text{"the"}, \text{"high"}, \text{"cost"})]$ $= [P(\text{"the"}   DT) * P(\text{"high"}   JJ) * P(\text{"cost"}   NN) * P(DT) * P(JJ   DT) * P(NN   JJ)] / \cancel{[P(\text{"the"}, \text{"high"}, \text{"cost"})]}$ $P(DT, NN, JJ   \text{"the"}, \text{"high"}, \text{"cost"}) = [P(\text{"the"}   DT) * P(\text{"high"}   NN) * P(\text{"cost"}   JJ) * P(DT) * P(NN   DT) * P(JJ   NN)] / \cancel{[P(\text{"the"}, \text{"high"}, \text{"cost"})]}$ $P(JJ, DT, NN   \text{"the"}, \text{"high"}, \text{"cost"}) = [P(\text{"the"}   JJ) * P(\text{"high"}   DT) * P(\text{"cost"}   NN) * P(JJ) * P(DT   JJ) * P(NN   DT)] / \cancel{[P(\text{"the"}, \text{"high"}, \text{"cost"})]}$ $[DT, JJ, NN] \rightarrow [P(\text{"the"}   DT) * P(\text{"high"}   JJ) * P(\text{"cost"}   NN) * P(DT) * P(JJ   DT) * P(NN   JJ)]$ $[DT, NN, JJ] \rightarrow [P(\text{"the"}   DT) * P(\text{"high"}   NN) * P(\text{"cost"}   JJ) * P(DT) * P(NN   DT) * P(JJ   NN)]$ $[JJ, DT, NN] \rightarrow [P(\text{"the"}   JJ) * P(\text{"high"}   DT) * P(\text{"cost"}   NN) * P(JJ) * P(DT   JJ) * P(NN   DT)]$
--	--

Let's analyse the process of stochastic parsing step by step.

The objective is to find the **most probable POS tag sequence** for the phrase: 'The high cost'. Simplifying assumption that only 2 possible tags exist: Determinant (DT), Adjective (JJ) and Noun (NN).

Each word in the sentence can be assigned any one of the three tags DT, JJ or NN. So, there could be  $3^3 = 27$  possible tag sequences (DT-DT-DT, DT-JJ-NN, DT-JJ-JJ, ....).

For simplicity, consider only these 3 possible tag sequences:

1. (DT, JJ, NN)
2. (DT, NN, JJ)
3. (JJ, DT, NN)

We need to find the maximum of [**P (tag sequence|observation sequence)**] among the possible tag sequences. Let's start with first tag sequence: (DT, JJ, NN)

$$P(DT, JJ, NN | The, high, cost) = \frac{P(The, high, cost | DT, JJ, NN) * P(DT, JJ, NN)}{P(The, high, cost)}$$

Expanding the terms on the right side by using the chain-rule:

$$P(DT, JJ, NN | the, high, cost) =$$

$$\frac{P(the | DT, JJ, NN) * P(high | the, DT, JJ, NN) * P(cost | the, high, DT, JJ, NN) * P(DT) * P(JJ | DT) * P(NN | DT, JJ)}{P(the, high, cost)}$$

We make an important assumption here called the **Markov assumption**. Briefly, a Markov process transitions from one 'state' to another with some probability. In this case, the states are the POS tags.



The Markov assumption states that the probability of a state depends only on the probability of the previous state leading to it. That implies, in a tag sequence (DT, JJ, NN), the probability that a word is tagged as NN depends only on the previous tag JJ and not on DT.

Another simplifying assumption we make is that the probability of a word  $w$  being assigned a tag  $t$  depends only on the tag  $t$  and not on any other tag. Thus, the probability  $P(\text{the}|\text{DT})$ , i.e. the probability that a word is 'the' given its tag is DT, depends only on DT and not on any of the other tags NN or JJ.

Simplifying the equation using these assumptions, we get:

$$P(DT, JJ, NN | The, high, cost) = \frac{P(the|DT) * P(high|JJ) * P(cost|NN) * P(DT) * P(JJ|DT) * P(NN|JJ)}{P(The, high, cost)}$$



Let's write similar equations for the other two candidate sequences:

$$P(DT, NN, JJ | The, high, cost) = \frac{P(the|DT) * P(high|NN) * P(cost|JJ) * P(DT) * P(NN|DT) * P(JJ|NN)}{P(The, high, cost)}$$

$$P(JJ, DT, NN | The, high, cost) = \frac{P(the|JJ) * P(high|DT) * P(cost|NN) * P(JJ) * P(DT|JJ) * P(NN|DT)}{P(The, high, cost)}$$

The denominator can simply be ignored since it is the same across all candidate sequences.

We can optimally arrive at a solution for getting the most probable tag sequence without calculating the probabilities for the 27 tag sequences. For now, let's solve some problems.

<p><b>Stochastic Modelling</b></p> <p>Select which of the following assumptions (both implicit and explicit) are made while doing stochastic parsing.</p> <p><input checked="" type="radio"/> The probability of tag for a given word is dependent upon the word and the previous tag in a sequence <span style="color: green;">✔</span> Correct</p> <p> Feedback : Markov assumption: the probability of a state depends only on the previous state</p> <p><input type="radio"/> The probability of next tag is dependent on past k tags, where k &gt; 1</p> <p><input type="radio"/> Both A and B</p>	<p><b>Stochastic modeling</b></p> <p>For the sequence of 3 words: 'The high cost' and three possible tags: (Noun, Adjective, Determinant), there are 27 possible tag sequences. What will be the generalized expression of n terms and t tags?</p> <p><input type="radio"/> n^t</p> <p><input type="radio"/> t^n <span style="color: green;">✔</span> Correct</p> <p> Feedback : t*t*t... (n times)</p>
--	--

**Stochastic modeling**

Which of the following are **incorrect** according to the Markov assumption? More than one options may be correct:

☐  $P(DT, Adj, N) = P(DT) \cdot P(Adj|DT) \cdot P(N|Adj)$

☒  $P(DT, Adj, Adj) = P(DT) \cdot P(Adj|DT) \cdot P(Adj|DT)$ 
✓ Correct

🔍 Feedback :

3rd term should be:  $P(Adj|Adj)$

☒  $P(DT, Adj, N) = P(DT) \cdot P(Adj|DT) \cdot P(N|Adj, DT)$ 
✓ Correct

🔍 Feedback :

3rd term should be:  $P(N|Adj)$ . Current state depends only on the previous state

☒  $P(DT, Adj, N) = P(DT) \cdot P(DT|Adj) \cdot P(Adj|N)$ 
✓ Correct

🔍 Feedback :

3rd term should be:  $P(N|Adj)$  and the 2nd term should be:  $P(Adj|DT)$ .

$P(\text{"the"}   DT)$	0.6		$(DT, Adj, N)$ the high cost			$(Adj, N, N)$ the high cost	
$P(Adj)$	0.2		$P(\text{"the"}   DT)$	0.6		$P(\text{"the"}   Adj)$	0.1
$P(\text{"the"}   Adj)$	0.1		$P(\text{"high"}   Adj)$	0.4		$P(\text{"high"}   N)$	0.2
$P(Adj   DT)$	0.3		$P(\text{"cost"}   N)$	0.3		$P(\text{"cost"}   N)$	0.3
$P(\text{"high"}   Adj)$	0.4		$P(DT)$	0.4		$P(Adj)$	0.2
$P(N   DT)$	0.4		$P(Adj   DT)$	0.3		$P(N   DT)$	0.4
$P(\text{"high"}   N)$	0.2		$P(N   Adj)$	0.5		$P(N   N)$	0.1
$P(N   Adj)$	0.5			0.00432			0.000048
$P(\text{"cost"}   N)$	0.3		$(DT, N, N)$ the high cost				
$P(N   N)$	0.1		$P(\text{"the"}   DT)$	0.6			
$P(DT)$	0.4		$P(\text{"high"}   N)$	0.2			
$P(Adj   Adj)$	0.2		$P(\text{"cost"}   N)$	0.3			
$P(Adj   N)$	0.1		$P(DT)$	0.4			
$P(DT   N)$	0.1		$P(N   DT)$	0.4			
			$P(N   N)$	0.1			
				0.000576			

## The Viterbi Heuristic

Previous segment had how to calculate the probability of a tag sequence given a sequence of words. The idea is to compute the probability of all possible tag sequences and assign the sequence having the maximum probability.

Although this approach can work in principle, it is computationally very expensive. With just three POS tags - DT, JJ, NN, tagging the sentence "The high cost", there are  $3^3=27$  possible tag sequences (each word can have three possible tags).

In general, for a sequence of  $n$  words and  $t$  tags, a total of  $t^n$  tag sequences are possible. The Penn Treebank dataset in NLTK itself has 36 POS tags, so for a sentence of length say 10, there are  $36^{10}$  possible tag sequences (that's about three thousand trillion!)

Clearly, computing trillions of probabilities to tag a 10-word sentence is impractical. Thus, we need to find a much more efficient approach to tagging.

One such approach is the **Viterbi heuristic**, also commonly known as the **Viterbi algorithm**.

<b>PENN TREEBANK</b> String = word1 word2 .... wordn $36 \times 36 \times 36 \dots = 36^n$	<b>POS TAGGING</b> Viterbi Heuristic  W: Words and T: Tags $[P(W_1   T_1) * P(W_2   T_2) * P(W_3   T_3) * P(T_1) * P(T_2   T_1) * P(T_3   T_2)]$ $= [P(W_1   T_1) * P(T_1)] * [P(W_2   T_2) * P(T_2   T_1)] * [P(W_3   T_3) * P(T_3   T_2)]$	
<b>VITERBI HEURISTIC</b>  Tag For "The" $P(DT start) * P("The" DT) = 0.4 * 0.6 = 0.24$ $P(NN start) * P("The" NN) = 0.1 * 0.2 = 0.02$ $P(JJ start) * P("The" JJ) = 0.2 * 0.1 = 0.02$ $P(VB start) * P("The" VB) = 0.3 * 0.1 = 0.03$  $P("The" JJ) = 0.1, P("high" JJ) = 0.4, P("cost" JJ) = 0.2,$ $P("The" NN) = 0.2, P("high" NN) = 0.2, P("cost" NN) = 0.3,$ $P("The" DT) = 0.6, P("high" DT) = 0.2, P("cost" DT) = 0.2,$ $P("The" VB) = 0.1, P("high" VB) = 0.2, P("cost" VB) = 0.3$		<b>VITERBI HEURISTIC</b>  Tag For "High" $P(DT DT) * P("high" DT) = 0.1 * 0.2 = 0.02$ $P(NN DT) * P("high" NN) = 0.4 * 0.2 = 0.08$ $P(JJ DT) * P("high" JJ) = 0.3 * 0.4 = 0.12$ $P(VB DT) * P("high" VB) = 0.2 * 0.1 = 0.02$
<b>Tag For "High"</b> $P(DT DT) * P("high" DT) = 0.1 * 0.2 = 0.02$ $P(NN DT) * P("high" NN) = 0.4 * 0.2 = 0.08$ $P(JJ DT) * P("high" JJ) = 0.3 * 0.4 = 0.12$ $P(VB DT) * P("high" VB) = 0.2 * 0.1 = 0.02$  <b>Tag For "Cost"</b> $P(DT JJ) * P("cost" DT) = 0.2 * 0.2 = 0.04$ $P(NN JJ) * P("cost" NN) = 0.5 * 0.3 = 0.15$ $P(JJ JJ) * P("cost" JJ) = 0.2 * 0.2 = 0.04$ $P(VB JJ) * P("cost" VB) = 0.1 * 0.3 = 0.03$		

Basic idea of **Viterbi algorithm** - given a list of observations (words)  $O_1, O_2, \dots, O_n$  to be tagged, rather than computing the probabilities of all possible tag sequences, assign tags sequentially, i.e. assign the most likely tag to each word using the previous tag.

Formally, assign tag  $T_j$  to each word  $O_i$  such that it maximises the likelihood:

$$P(T_j|O_i) = P(O_i|T_j) * P(T_j|T_{j-1}),$$

where  $T_{j-1}$  is the tag assigned to the previous word. According to the Markov assumption, the probability of a tag  $T_j$  is assumed to be **dependent only on the previous tag**  $T_{j-1}$ , and hence the term  $P(T_j|T_{j-1})$ .

In other words, assign tags sequentially such that the tag assigned to every word  $O_i$  maximises the likelihood  $P(T_j|O_i)$  locally, i.e. it is assumed to be dependent only on the current word and the previous tag. This algorithm does not guarantee that the resulting tag sequence will be the most likely sequence, but by making these simplifying assumptions, it reduces the computational time manifold.

This is also why it is called a **greedy algorithm** - it assigns the tag that is most likely *at every word*, rather than looking for the overall most likely sequence.

Compare computational costs of two tagging algorithms - the brute force algorithm and the Viterbi algorithm.

HIDDEN MARKOV MODEL	Viterbi Heuristic <sup>UpG</sup>
	<p><b>Important correction:</b> The professor mentioned that the order of computations using the Viterbi algorithm is <math>O(nS)</math>, though it is <math>O(nS^2)</math>.</p> <p>At each <math>n</math> observations, algorithm does the following computations:</p> <ol style="list-style-type: none"> <li>1. Compute <math>S</math> state probabilities corresponding to <math>S</math> candidate tags, thus doing <math>O(S)</math> computations and</li> <li>2. <b>Compute the maximum</b> of the <math>S</math> probabilities, whose order is again <math>O(S)</math>.</li> </ol>

[Read here](#) to understand why the time complexity of computing the maximum of  $n$  numbers is  $O(n)$ .

After familiarity with the basic Viterbi algorithm, next is Markov processes and Hidden Markov Models more formally. Later segments cover a POS tagging algorithm using the Viterbi heuristic.

### Viterbi Heuristic

How does Viterbi Heuristic help in reducing the number of calculations in getting tag sequence?

- ☐ For a possible tag sequence, Viterbi sums over the probabilities of all possible tags that could generate the observation sequence

- ☒ Viterbi follows a greedy approach by maximizing the likelihood of the tag for a given observation

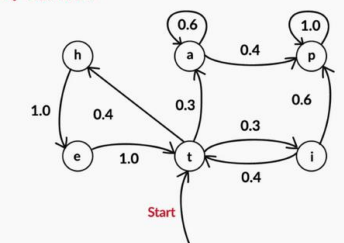
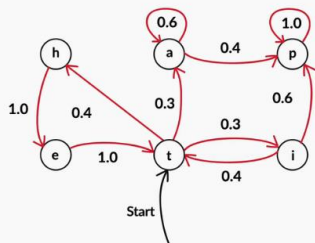
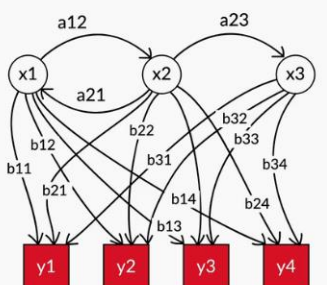
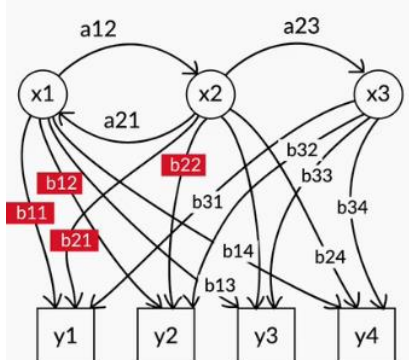
💡 Feedback :

*Viterbi for each observation maximises the likelihood of the tag*

- ☐ Viterbi Heuristic selects random tag sequences from all possible brute force calculations
- ☐ None of the above

## Markov Chain and HMM

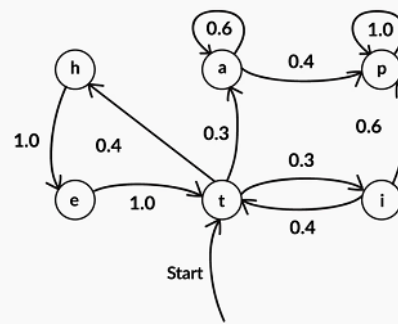
**Markov models** are probabilistic (or stochastic) models that were developed to model sequential processes. In a Markov process, it is usually assumed that the probability of each event (or state) depends only on the probability of the previous event. This simplifying assumption is a special case which is known as the Markovian, one-Markov and the first-order Markov assumption.

<p><b>HIDDEN MARKOV MODEL (HMM)</b></p> <p><b>Markov Assumption</b></p> <p>A sequence of random variables (<math>X_1, X_2, \dots, X_T</math>) are said to form a Markov process iff:</p> $P(X_{i+1} = s   X_1, X_2, \dots, X_i) = P(X_{i+1} = s   X_i)$	<p><b>HMM</b></p> <p>Transition Probability</p> <p><math>S_0</math> = Start State</p> 	<p><b>HMM</b></p> <p>Transition Probability</p> <p><math>S_0</math> = Start State</p> 
<p><b>POS TAGGING PROBLEM STATEMENT</b></p> <p>Markov States Sequence: <math>? \xrightarrow{P} ? \xrightarrow{P} ?</math></p> <p>Emission Sequence: The high cost</p>	<p><b>HIDDEN MARKOV MODEL (HMM)</b></p> <p>Probability of next state depends only on the previous state</p>	<p><b>HMM</b></p> <p>Emission Probability</p> 
<p><b>HMM</b></p> <p>Emission Probability</p> 	<p><b>HMM</b></p> <p>Emission Probability</p> <p>Markov States Sequence: <math>? \xrightarrow{P} ? \xrightarrow{P} ?</math></p> <p>Emission Sequence: The high cost</p> <ol style="list-style-type: none"><li>1. States: <math>x_1, x_2, x_3</math></li><li>2. Transition probability: <math>a_{12}, a_{21}, a_{23}</math></li><li>3. Output: <math>y_1, y_2, y_3, y_4</math></li><li>4. Emission probability: <math>b_{11}, b_{12}, b_{21}, b_{12}, b_{13} \dots</math></li><li>5. Sequence of words: <math>y_1, y_2, y_3, y_4</math></li><li>6. POS tags: <math>x_1, x_2, x_3</math></li></ol>	

Let's summarise the theory of Markov processes and HMMs.

A Markov chain is used to represent a process which performs a transition from one state to other. This transition makes an assumption that the probability of transitioning to the next state is dependent solely on the current state.

$S_0$  = Start State



Here, 'a', 'p', 'i', 't', 'e', 'h' are the **states** and the numbers mentioned on the edges are **transition probabilities**. For e.g. the probabilities of transitioning from the state 't' to the states 'i', 'a' and 'h' are 0.3, 0.3, 0.4 respectively.

The **start state** is a special state which represents the initial state of the process (e.g. the start of a sentence).

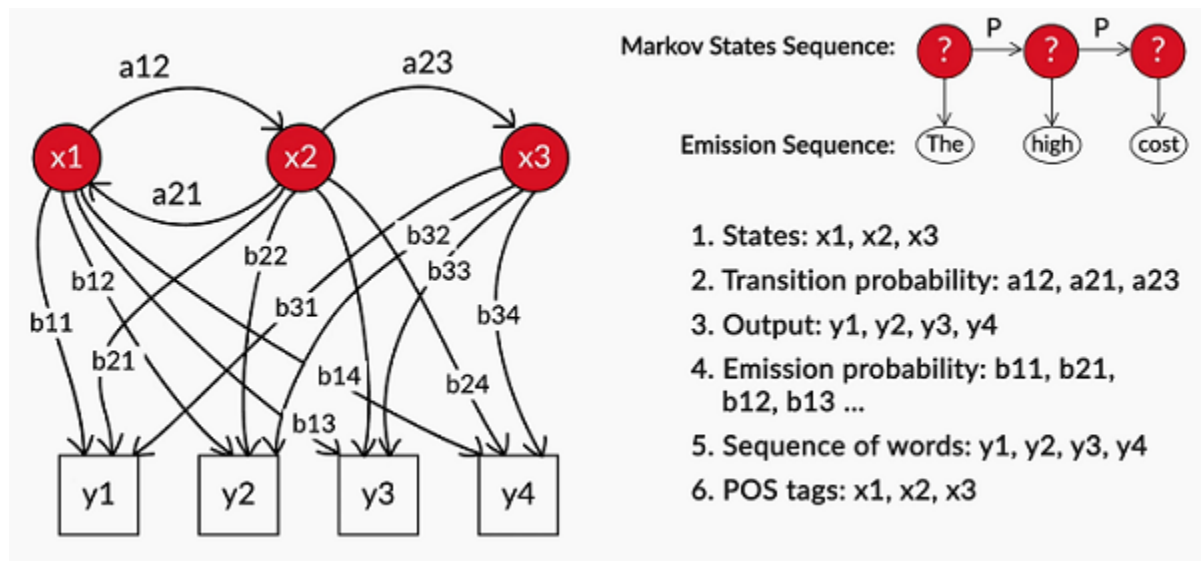
**Markov processes** are commonly used to model sequential data, such as text and speech. To build an application which predicts the next word in a sentence, represent each word in a sentence as a state. The transition probabilities (which can be learnt from some corpus, more on that later) would represent the probability that the process moves from the current word to the next word. For e.g. the transition probability from the state 'San' to 'Francisco' will be higher than to the state 'Delhi'.

The **Hidden Markov Model (HMM)** is an extension to the Markov process which is used to model phenomena where the **states are hidden** (or latent) and they **emit observations**. For example, in a speech recognition system (a speech-to-text converter), the states represent the actual text words to be predicted, but no direct observation (i.e. the states are hidden). Rather, observe the speech (audio) signals corresponding to each word, and infer the states using the observations.

Similarly, in POS tagging, observations are the words in a sentence, while the POS tags themselves are hidden. Model the POS tagging task as an HMM with the hidden states representing POS tags which emit observations, i.e. words.

The hidden states **emit observations** with a certain probability. Therefore, along with the transition and initial state probabilities, Hidden Markov Models also have **emission probabilities** which represent the probability that an observation is emitted by a particular state.

The figure below illustrates the emission and transition probabilities for a hidden Markov process having three hidden states and four observations.



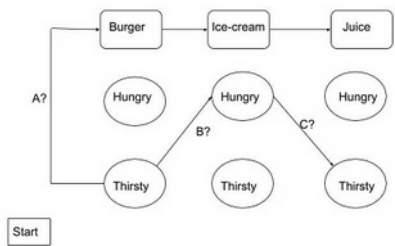
## HMM

Previous segment used the transition and the emission probabilities for finding the most probable tag sequence for the sentence "The high cost". The probabilities  $P(NN|JJ)$ ,  $P(JJ|DT)$  etc. are transition probabilities, while the  $P(\text{high}|JJ)$ ,  $P(\text{cost}|NN)$  etc. are the emission probabilities.

<p><b>HMM</b></p> <p>What are the parameters required for defining an HMM? (Multiple options are correct)</p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <input checked="" type="checkbox"/> Emission and Transition Probability         </div> <p><b>Feedback:</b> Emission and state transition probabilities are required for defining an HMM</p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <input checked="" type="checkbox"/> Sequence of words         </div> <p><b>Feedback:</b> The sequence of words helps in solving a type of problem like POS tagging. But it is not required for defining HMM model parameters</p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <input type="checkbox"/> POS tags         </div> <div style="border: 1px solid #ccc; padding: 5px;"> <input type="checkbox"/> Initial state and initial state distribution         </div> <p><b>Feedback:</b> Initial state probabilities are also required to determine most likely tag with which a sentence begins.</p>	<p><b>HMM</b></p> <p>What is 'hidden' in hidden Markov model?</p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <input type="radio"/> Observations         </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <input checked="" type="radio"/> States         </div> <p><b>Feedback:</b> Correct. States (or tags) are hidden.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <input type="radio"/> Initial state probabilities         </div> <div style="border: 1px solid #ccc; padding: 5px;"> <input type="radio"/> Transition and emission probability         </div>
---	--



Ram eats a burger, an ice-cream, and drinks a juice in some sequence. He could be either in the hungry or the thirsty state. A, B, C represent the transition/emission probabilities. Can you select which option correctly represents A, B, and C?



☒ A -  $P(\text{Burger}|\text{thirsty})$ , B -  $P(\text{hungry}|\text{thirsty})$ , C -  $P(\text{thirsty}|\text{hungry})$

✓ Correct

Feedback:

A is emission probability, B and C are transition probabilities

☐ A -  $P(\text{thirsty}|\text{Burger})$ , B -  $P(\text{thirsty}|\text{hungry})$ , C -  $P(\text{hungry}|\text{thirsty})$

☐ A -  $P(\text{thirsty}|\text{hungry})$ , B -  $P(\text{hungry}|\text{thirsty})$ , C -  $P(\text{Burger}|\text{thirsty})$

☐ A -  $P(\text{thirsty}|\text{Burger})$ , B -  $P(\text{hungry}|\text{thirsty})$ , C -  $P(\text{thirsty}|\text{hungry})$

#### HMM

Select the option which correctly defines the emission probability of a Hidden Markov Model:

☒ The probability with which an observation is emitted by a given latent state

Feedback:

Emission probability =  $P(\text{observation}|\text{state})$ .

☐ The probability with which a state is emitted by an observation

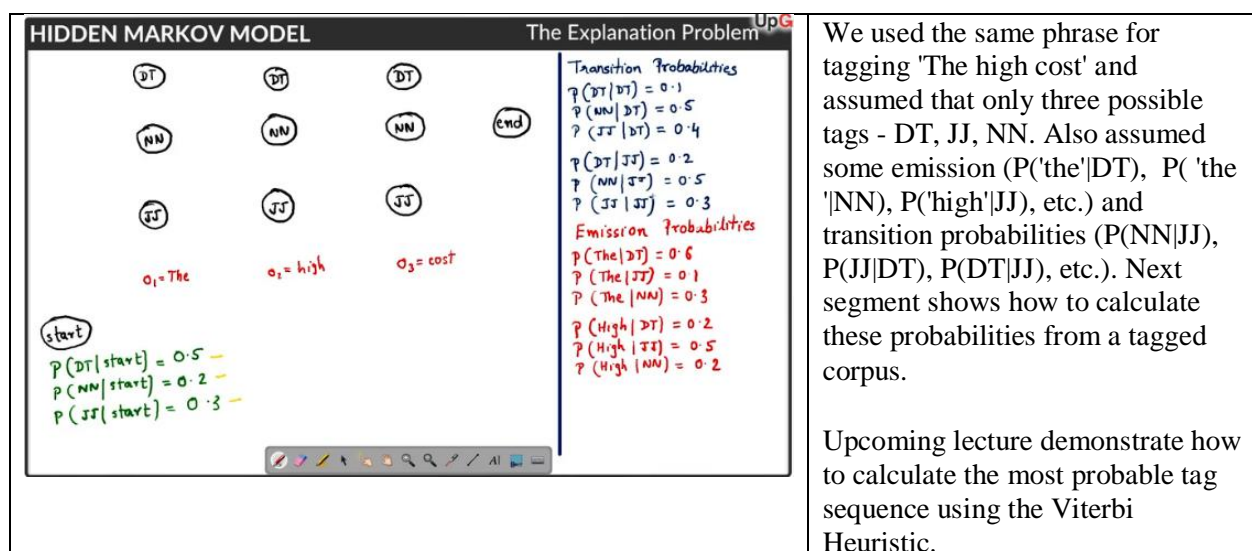
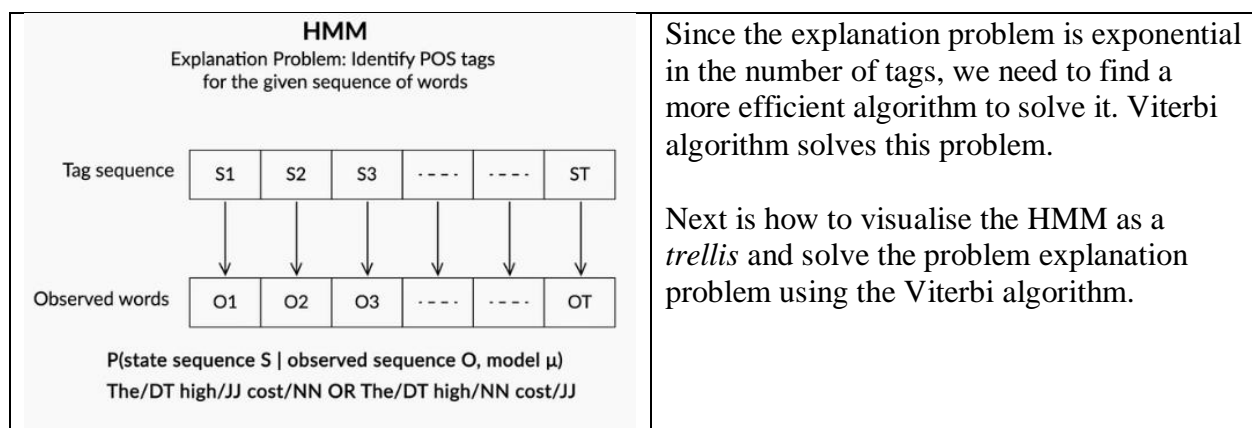
☐ The probability with which a state transitions into another state

☐ None of the above

## Explanation Problem

POS tagging problem can be modelled using an HMM. Sequence of words are the observations while the POS tags are the hidden states. Also, the HMM is represented by its initial state probabilities (i.e. the probability of transition into any state from the initial state), the transition and the emission probabilities.

These probabilities are usually learnt from a training corpus. For now, assume that already learnt these probabilities. Then, the **explanation problem**, also called the **decoding problem**, is as follows: Given a sequence of words/observations and an HMM model (i.e. transition, emission and start state probabilities), find the tag/state sequence which maximises the probability of observing the observed words.



<p><b>HIDDEN MARKOV MODEL</b> <span style="float: right;">The Explanation Problem <sup>UpG</sup></span></p> <p>start</p> $P(DT start) = 0.5 \times 0.6 = 0.3$ $P(NN start) = 0.2 \times 0.3 = 0.06$ $P(JJ start) = 0.3 \times 0.1 = 0.03$ <p>Transition Probabilities</p> $P(DT DT) = 0.1$ $P(NN DT) = 0.5$ $P(JJ DT) = 0.4$ $P(DT JJ) = 0.2$ $P(NN JJ) = 0.5$ $P(JJ JJ) = 0.3$ <p>Emission Probabilities</p> $P(The DT) = 0.6$ $P(The JJ) = 0.1$ $P(The NN) = 0.3$ $P(High DT) = 0.2$ $P(High JJ) = 0.5$ $P(High NN) = 0.2$	<p>Start will be given DT since The as DT has highest probability</p>
<p><b>HIDDEN MARKOV MODEL</b> <span style="float: right;">The Explanation Problem <sup>UpG</sup></span></p> <p>start</p> $P(DT start) = 0.5 \times 0.6 = 0.3$ $P(NN start) = 0.2 \times 0.3 = 0.06$ $P(JJ start) = 0.3 \times 0.1 = 0.03$ <p>Transition Probabilities</p> $P(DT DT) = 0.1 \times 0.2 = 0.02$ $P(NN DT) = 0.5 \times 0.2 = 0.1$ $P(JJ DT) = 0.4 \times 0.5 = 0.2$ $P(DT JJ) = 0.2$ $P(NN JJ) = 0.5$ $P(JJ JJ) = 0.3$ <p>Emission Probabilities</p> $P(The DT) = 0.6$ $P(The JJ) = 0.1$ $P(The NN) = 0.3$ $P(High DT) = 0.2$ $P(High JJ) = 0.5$ $P(High NN) = 0.2$	<p>Here JJ wins with 0.2</p>
<p><b>HIDDEN MARKOV MODEL</b> <span style="float: right;">The Explanation Problem <sup>UpG</sup></span></p> <p>start</p> $P(DT start) = 0.5 \times 0.6 = 0.3$ $P(NN start) = 0.2 \times 0.3 = 0.06$ $P(JJ start) = 0.3 \times 0.1 = 0.03$ <p>Transition Probabilities</p> $P(DT DT) = 0.1 \times 0.2 = 0.02$ $P(NN DT) = 0.5 \times 0.2 = 0.1$ $P(JJ DT) = 0.4 \times 0.5 = 0.2$ $P(DT JJ) = 0.2$ $P(NN JJ) = 0.5$ $P(JJ JJ) = 0.3$ <p>Emission Probabilities</p> $P(The DT) = 0.6$ $P(The JJ) = 0.1$ $P(The NN) = 0.3$ $P(High DT) = 0.2$ $P(High JJ) = 0.5$ $P(High NN) = 0.2$	<p>It is like a trellis where tag from one level gets linked with tag from another level and so on till sequence ends.</p>

Given a Hidden Markov Model defined by the initial state, emission, and the transition probabilities, assign a tag  $T_j$  to an observation  $O_i$  such that it maximises the likelihood:

$$P(T_j|O_i) = P(O_i|T_j) * P(T_j|T_{j-1})$$

Viterbi algorithm, is a **dynamic programming** algorithm. It break down a complex problem into subproblems and solve each subproblem optimally.

Until now, we assumed some initial state, transition and emission probabilities and used them to compute the optimal tag sequence. But how to learn these probabilities? This task is done using some tagged corpus, such as the Penn Treebank, and is called **the learning problem**.

Next segments solves the learning problem and write an algorithm in Python to train own HMM model using the Viterbi algorithm.

**Explanation Problem**

Consider the sentence: "The man slept" and only 3 possible tags: Noun, Determinant, and Verb. Below is the trellis for the same. Find the probability of tag sequence  $P(DT, NN \mid \text{"the", "man"})$ .

$P(DT DT) = 0.3$	$P(DT NN) = 0.2$	$P(DT VB) = 0.3$
$P(NN DT) = 0.4$	$P(NN NN) = 0.3$	$P(NN VB) = 0.4$
$P(VB DT) = 0.3$	$P(VB NN) = 0.5$	$P(VB VB) = 0.3$

☐ 0.0432  
☐ 0.0324  
☒ 0.0576  
☐ 0.0144

**Feedback:**

$P(DT, NN \mid \text{"the", "man"}) = P(\text{"the"}|DT) * P(DT|start) * P(\text{"man"}|NN) * P(NN|DT)$

## Viterbi

Dynamic programming is a very powerful technique to solve a particular class of problems. The idea is very simple, If you have solved a problem with the given input, then save the result for future reference, so as to avoid solving the same problem again.. shortly 'Remember your Past' :). Viterbi also does the same. For any given sequence of words, it calculates the most probable tag for the word and also keeps a backpointer for keeping track of the path of the tags that led to current tag. Which of the following algorithm is a dynamic programming variant, which resembles Viterbi?

☒ Minimum Edit distance



Correct

 Feedback :

*Minimum edit distance also finds the path which leads to minimum possible edits. At each point, it also keeps a track how it reached that step*

☐ Bag of words model

☐ Naive Bayes

## Additional References:

- Likelihood problem of predicting the next possible word can be solved using forward algorithm. You can read about it in more detail from this link:  
<https://web.stanford.edu/~jurafsky/slp3/9.pdf> (Refer to the section 9.3)

## Learning HMM Model Parameters

Compute the emission & transition probabilities from a tagged corpus. This process of learning the probabilities from a tagged corpus is called **training an HMM** model.

<p style="text-align: right;">UpGrad</p> <p style="text-align: center;"><b>HMM</b> Model Learning</p> <ol style="list-style-type: none"> <li>1. I/pronoun must/verb help/verb her/pronoun at/preposition any/determiner <b>cost/noun</b></li> <li>2. The/determiner <b>cost/noun</b> of/preposition living/noun is/verb rising/verb</li> <li>3. This/determiner computer/noun did/verb not/adverb <b>cost/verb</b> much/adverb</li> <li>4. I/pronoun did/verb not/adverb expect/verb it/pronoun to <b>cost/verb</b> so/adverb much/adjective</li> <li>5. How/adverb much/adverb does/verb it/pronoun <b>cost/verb</b> to get/verb to Boston/noun?</li> <li>6. We/pronoun shared/verb the/determiner <b>cost/noun</b> of/preposition the/determiner meal/noun</li> <li>7. Avoid/verb it/pronoun at/preposition all/determiner <b>cost/noun</b></li> </ol>	
<p style="text-align: right;">UpGrad</p> <p style="text-align: center;"><b>HMM</b> Model Learning</p> <ol style="list-style-type: none"> <li>1. I/pronoun must/verb help/verb her/pronoun at/preposition any/determiner <span style="border: 1px solid red;">cost</span>/noun</li> <li>2. The/determiner <span style="border: 1px solid red;">cost</span>/noun of/preposition living/noun is/verb rising/verb</li> <li>3. This/determiner computer/noun did/verb not/adverb <span style="border: 1px solid red;">cost</span>/verb much/adverb</li> <li>4. I/pronoun did/verb not/adverb expect/verb it/pronoun to <span style="border: 1px solid red;">cost</span> verb so/adverb much/adjective</li> <li>5. How/adverb much/adverb does/verb it/pronoun <span style="border: 1px solid red;">cost</span> verb to get/verb to Boston/noun?</li> <li>6. We/pronoun shared/verb the/determiner <span style="border: 1px solid red;">cost</span>/noun of/preposition the/determiner meal/noun</li> <li>7. Avoid/verb it/pronoun at/preposition all/determiner <span style="border: 1px solid red;">cost</span>/noun</li> </ol>	
<p style="text-align: center;"><b>HMM</b> Model Learning</p> <p>Emission Probability = <math>\frac{\#Cost/Noun}{\#Noun} = \frac{\text{Number of times the term "Cost" has been tagged as "Noun"}}{\text{Number of times the tag "Noun" appears}} = \frac{4}{8} = 0.5</math></p> <p>Transition Probability = <math>\frac{\#Pronoun \text{ followed by Verb}}{\#Pronoun \text{ tags in the corpus}} = \frac{\text{Number of times the tag "Pronoun" is followed by "Verb"}}{\text{Number of times the tag "Pronoun" appears}}</math></p> <p style="text-align: center;"><math>= \frac{4}{7}</math></p>	

To summarise, the emission and the transition probabilities can be learnt as follows:

**Emission Probability** of a word 'w' for tag 't':

$$P(w|t) = \text{Number of times } w \text{ has been tagged } t / \text{Number of times } t \text{ appears}$$

**Transition Probability** of tag t1 followed by tag t2:

$$P(t2|t1) = \text{Number of times } t1 \text{ is followed by tag } t2 / \text{Number of times } t1 \text{ appears}$$

Let's assume you have the following POS tagged corpus. Now answer the questions in the quiz section.  
(Text is in form of **word/tag**)

[Twitter/NN is/VB the/DT most/JJ open/JJ social/JJ media/NN platform/NN, which/WDT is/VB partly/RB why/WRB it/PRP is/VB used/VB by/IN so/RB many/JJ politicians/NN, celebrities/NN, journalists/NN, technocrats/NN, and/CC experts/NN working/VB on/IN pacy/JJ topics/NN.

As/IN we/PRP learned/VB over/IN the/DT past/JJ year/NN, openness/NN of/IN Twitter/NN was/VB exploited/VB by/IN adversarial/JJ governments/NN trying/VB to/TO influence/VB elections/NN.

Twitter/NN is/VB marketing/VB itself/PRP as/IN /DT news/NN platform/NN.]

The table below summarises the counts of the above text:

	#		#
NN	17	VB	12
JJ	7	DT	3
Twitter appearing as NN	3	Was appearing as VB	1
JJ followed by NN	5	Exploited appearing as VB	1
NN followed by VB	4	VB followed by VB	3

#### Model Parameters

Select the correct option for  $P("Twitter"|NN)$ :

☐ 0.176

🔍 Feedback :

$P("Twitter"|NN) = \text{Number of times 'Twitter' appearing as Noun} / \text{Number of Nouns} (3/17)$

☐ 0.118

☐ 0.059

☒ 0.167

🔍 Feedback :

$P("Twitter"|NN) = \text{Number of times 'Twitter' appearing as Noun} / \text{Number of Nouns}$

#### Model Parameters

Select the correct option for  $P(NN|JJ)$ :

☐ 0.411

☐ 0.278

☒ 0.714

🔍 Feedback :

$P(NN|JJ) = \text{number of times adjective is followed by a noun} / \text{Number of adjectives} (5/7)$

☐ 0.389

#### Model Parameters

Select the correct probability for the tag sequence  $P(NN|VB|VB)$  for the sentence: "Twitter was exploited".

(Consider  $P(NN|start) = P(N|;) = 0.333$ )

☐  $4.88 * 10^{-5}$

☐  $1.22 * 10^{-5}$

☒  $7.18 * 10^{-5}$

✗ Incorrect

🔍 Feedback :

$P(("Twitter"|NN) * P(NN|start) * ("was"|VB) * P(VB|NN) * ("exploited"|VB) * P(VB|VB)$

☐  $2.40 * 10^{-5}$

✓ Correct

🔍 Feedback :

$P(("Twitter"|NN) * P(NN|start) * ("was"|VB) * P(VB|NN) * ("exploited"|VB) * P(VB|VB)$



## HMM and the Viterbi Algorithm: Pseudocode

Basic idea behind the two main problems in building an HMM for POS tagging - **the learning problem** (learning the probabilities) and **the explanation problem** (solved using the Viterbi algorithm). Next segment builds an HMM using the Penn Treebank as the training corpus.

Before that, reiterate the important concepts and understand the pseudocode of this program.

The Penn Treebank is a manual of around a million words taken from 1989 Wall Street Journal's articles. This manual is available in NLTK toolkit of Python. Explore this data by importing NLTK and then run the following code:

```
wsj = list(nltk.corpus.treebank.tagged_sents())
```

The 'wsj' object is a list of 'list of tuples'. Each tuple is in the form of (word, POS tag). Already performed EDA on the Treebank data while building the lexicon and rule-based taggers.

### Learning the HMM Model Parameters

Let's start with training the HMM to learn the probabilities. For this, calculate the **transition probability** (from one state to another), the **emission probability** (of a state emitting a word), and the **initial state probabilities** (of a state appearing at the start of a sentence).

#### Transition Probability

Transition probability is the probability of moving from one state to another.

<p><b>Training an HMM</b></p> <p>To calculate the transition probability i.e. the probability that tag t1 will transition to tag t2, which of following options is correct for calculating numerator and denominator of the transition probability</p> <p><input type="radio"/> Numerator - the number of times t1 existed in the corpus</p> <p>Denominator - the number of times t1 is followed by t1</p> <p><input checked="" type="radio"/> Numerator - the number of times t1 is followed by t2 <span style="color: green;">✓</span> Correct</p> <p>Denominator - the number of times t1 appeared in the corpus</p> <p><b>Feedback :</b></p> <p>Transition Probability: out of the total occurrences of t1, you want to know, how many of them transitioned to t2</p> <p><input type="radio"/> Numerator - the number of times a word is tagged t1</p> <p>Denominator - the number of times t1 appeared in the corpus</p>	<p>Thus, we compute the transition probabilities in a loop to get the probabilities for all possible POS tag pairs <math>P(t_j t_i)P(t_j t_i)</math>.</p>
---	---

## Emission Probability

Next, we calculate the **emission probabilities**, i.e. the probability that a tag 't' will 'emit' a word w.

<p><b>Emission Probabilities</b></p> <p>To calculate the emission probability i.e. the probability that the tag t will emit the word w, which of following options correctly represents the numerator and the denominator?</p> <p><input checked="" type="radio"/> Numerator: the number of times the word w is tagged t <span style="float: right;">✔ Correct</span></p> <p>Denominator: the number of times tag t appears in the corpus</p> <p><b>Feedback:</b></p> <p><i>Emission probability: Out of the total occurrences of t1, you want to know, how many of them omit word w</i></p> <p><input type="radio"/> Numerator: the number of times the word w appears in the corpus</p> <p>Denominator: the number of times tag t appears in the corpus</p> <p><input type="radio"/> Numerator: the number of times the tag t is associated with the word w</p> <p>Denominator: the number of times w appears in the corpus</p> <p><input type="radio"/> Numerator: the number of times word is tagged t</p> <p>Denominator: the number of times w appears in the corpus</p>	<p>We can compute the emission probabilities for all possible word-tag pairs in the corpus. Also, we can infer the start probabilities, i.e. <math>P(t   \text{start})P(t   \text{start})</math> for all possible tags t by computing the number of times the tag t appears at the start of a sentence.</p>
--	---

## The Explanation/Decoding Problem: Viterbi Algorithm

After learning the model parameters, we need to find the best possible state (tag) sequence for each given sentence. We'll use the Viterbi algorithm - for every word w in the sentence, a tag t is assigned to w such that it maximises the likelihood of the occurrence of  $P(\text{tag}|\text{word})$ .

$$\begin{aligned} P(\text{tag}|\text{word}) &= P(\text{word}|\text{tag}) * P(\text{tag}|\text{previous tag}) \\ &= \text{Emission probability} * \text{Transition probability} \end{aligned}$$

In other words, we assign the tag t to the word w which has the max  $P(\text{tag}|\text{word})$ .

We'll keep on storing the assigned tags and words as a list of tuples. As we move to the next word in the list, each tag to be assigned will use the tag of the previous word. **Trellis** is quite useful to visualise the algorithm.

For the start word of a sentence, which is a 'boundary case', the sentence terminator of the previous tag can be used to calculate the initial state probabilities.

This forms the basis of developing a POS tagger using the Viterbi heuristic. In the next segment, we'll see how to implement this pseudocode in Python.

## HMM & the Viterbi Algorithm: Python Implementation

Build an HMM POS tagger using the Penn Treebank dataset as the training corpus.

Do the following in Python:

1. Conduct exploratory analysis on a tagged corpus
2. Sample the data into 70:30 train-test sets
3. Train an HMM model using the tagged corpus:
  1. Calculating the emission probabilities  $P(w_i|t_j)$
  2. Calculating the transition probabilities  $P(t_i|t_{i-1})$
4. Write the Viterbi algorithm to POS tag sequence of words (sentence)
5. Evaluate the model predictions against the ground truth

How to compute the emission, transition and the initial state probabilities ( $P(\text{tag} | \text{start})$ ). A sentence can end with either of the three terms '.', '?' or '!'. They are all called sentence terminators and are tagged as '.'. Thus,  $P(\text{tag} | \text{start})$  is equivalent to  $P(\text{tag} | '.')$ .

Next, compute the transition probabilities matrix and finally use the emission and the transition probabilities to write the Viterbi algorithm for tagging a given sequence of words.

To summarise, the Viterbi algorithm works as follows:

1. For each word, compute the  $P(\text{tag} | \text{word})$  for each tag in the tag set and then assign the tag having the max  $P(\text{tag} | \text{word})$ .
2.  $P(\text{tag} | \text{word}) = (\text{emission probability of the word-tag pair}) * (\text{transition probability from the previous tag})$ .
3. As we move along each word, we keep on storing the assigned tags in a list. As we progress further down the sequence, each word uses the tag of the previous token to compute the transition probabilities.

<p>POS Tagging, HMMs, Viterbi</p> <p>Let's learn how to do POS tagging by Viterbi Heuristic using tagged Treebank corpus. Before going through the code, let's first understand the pseudo-code for the same.</p> <ol style="list-style-type: none"><li>1. Tagged Treebank corpus is available (Sample data to training and test data set)<ul style="list-style-type: none"><li>o Basic text and structure exploration</li></ul></li><li>2. Creating HMM model on the tagged data set.<ul style="list-style-type: none"><li>o Calculating Emission Probability: <math>P(\text{observation}   \text{state})</math></li><li>o Calculating Transition Probability: <math>P(\text{state2}   \text{state1})</math></li></ul></li><li>3. Developing algorithm for Viterbi Heuristic</li><li>4. Checking accuracy on the test data set</li></ol>	<pre>#Importing libraries import nltk, re, pprint import numpy as np import pandas as pd import requests import matplotlib.pyplot as plt import seaborn as sns import pprint, time import random from sklearn.model_selection import train_test_split from nltk.tokenize import word_tokenize</pre>
---	---

<p><b>2. POS Tagging Algorithm - HMM</b></p> <p>We'll use the HMM algorithm to tag the words. Given a sequence of words to be tagged, the task is to assign the most probable tag to the word.</p> <p>In other words, to every word <math>w</math>, assign the tag <math>t</math> that maximises the likelihood <math>P(t/w)</math>. Since <math>P(t/w) = P(w/t) \cdot P(t) / P(w)</math>, after ignoring <math>P(w)</math>, we have to compute <math>P(w/t)</math> and <math>P(t)</math>.</p> <p><math>P(w/t)</math> is basically the probability that given a tag (say NN), what is the probability of it being <math>w</math> (say 'building'). This can be computed by computing the fraction of all NNs which are equal to <math>w</math>, i.e.</p> <p><math>P(w/t) = \text{count}(w, t) / \text{count}(t)</math>.</p> <p>The term <math>P(t)</math> is the probability of tag <math>t</math>, and in a tagging task, we assume that a tag will depend only on the previous tag. In other words, the probability of a tag being NN will depend only on the previous tag <math>t(n-1)</math>. So for e.g. if <math>t(n-1)</math> is a JJ, then <math>t(n)</math> is likely to be an NN since adjectives often precede a noun (blue coat, tall building etc.).</p> <p>Given the penn treebank tagged dataset, we can compute the two terms <math>P(w/t)</math> and <math>P(t)</math> and store them in two large matrices. The matrix of <math>P(w/t)</math> will be sparse, since each word will not be seen with most tags ever, and those terms will thus be zero.</p>	<p><b>Emission Probabilities</b></p> <pre># compute word given tag: Emission Probability def word_given_tag(word, tag, train_bag = train_tagged_words):     tag_list = [pair for pair in train_bag if pair[1]==tag]     count_tag = len(tag_list)     w_given_tag_list = [pair[0] for pair in tag_list if pair[0]==word]     count_w_given_tag = len(w_given_tag_list)      return (count_w_given_tag, count_tag)</pre> <p><b>Transition Probabilities</b></p> <p># compute tag given tag: tag2(<math>t_2</math>) given tag1 (<math>t_1</math>), i.e. Transition Probability</p> <pre>def t2_given_t1(t2, t1, train_bag = train_tagged_words):     tags = [pair[1] for pair in train_bag]     count_t1 = len([t for t in tags if t==t1])     count_t2_t1 = 0     for index in range(len(tags)-1):         if tags[index]==t1 and tags[index+1] == t2:             count_t2_t1 += 1     return (count_t2_t1, count_t1)</pre>
<pre># creating t x t transition matrix of tags # each column is t2, each row is t1 # thus M(i, j) represents P(tj given ti)  tags_matrix = np.zeros((len(T), len(T)), dtype='float32') for i, t1 in enumerate(list(T)):     for j, t2 in enumerate(list(T)):         tags_matrix[i, j] = t2_given_t1(t2, t1)[0]/t2_given_t1(t2, t1)[1]</pre>	<p><b>3. Viterbi Algorithm</b></p> <p>Let's now use the computed probabilities <math>P(w, \text{tag})</math> and <math>P(t_2, t_1)</math> to assign tags to each word in the document. We'll run through each word <math>w</math> and compute <math>P(\text{tag}/w) = P(w/\text{tag}) \cdot P(\text{tag})</math> for each tag in the tag set, and then assign the tag having the max <math>P(\text{tag}/w)</math>.</p> <p>We'll store the assigned tags in a list of tuples, similar to the list 'train_tagged_words'. Each tuple will be a (token, assigned_tag). As we progress further in the list, each tag to be assigned will use the tag of the previous token.</p> <p>Note: <math>P(\text{tag} \text{start}) = P(\text{tag} \text{'.'})</math></p>
<pre># Viterbi Heuristic def Viterbi(words, train_bag = train_tagged_words):     state = []     T = list(set([pair[1] for pair in train_bag]))</pre>	<pre>## Testing sentence_test = ('Twitter is the best networking social site. Man is a social animal.')</pre>

<pre> for key, word in enumerate(words):     #initialise list of probability column for a given     observation     p = []     for tag in T:         if key == 0:             transition_p = tags_df.loc['.', tag]         else:             transition_p = tags_df.loc[state[-1], tag]          # compute emission and state probabilities         emission_p = word_given_tag(words[key], tag)[0]/word_given_tag(words[key], tag)[1]         state_probability = emission_p * transition_p         p.append(state_probability)      pmax = max(p)     # getting state for which probability is maximum     state_max = T[p.index(pmax)]     state.append(state_max) return list(zip(words, state)) </pre>	<pre> 'Data science is an emerging field. Data science jobs are high in demand.') words = word_tokenize(sentence_test)  start = time.time() tagged_seq = Viterbi(words) end = time.time() difference = end-start </pre>
---	---

### Viterbi Algorithm

The following code snippet is from 'Viterbi' function of Jupyter notebook

```

for tag in T:
    if key == 0:
        transition_p = tags_df.loc['.', tag]
    else:
        transition_p = tags_df.loc[state[-1], tag]

```

Select the option which best describes this step

- ☐ Emission probability is calculated for each tag
- ☐ For each tag, this step is calculating the  $P(\text{tag}|\text{word})$

☒ Transition probability is calculated for each tag. For the first word of a sentence, it is computing  $P(\text{tag}|\cdot)$  and for the rest it is computing  $P(\text{tag}|\text{previous\_tag})$

#### Feedback :

Transition probabilities are calculated for the tag. And tag for the first word is calculated taking '' as the previous tag

Let's evaluate the accuracy of the Viterbi algorithm on the test dataset. Predicting the tags of the entire test-set takes about 3-4 hours. So, we'll compute the tagging accuracy on a few randomly chosen sentences.

The Viterbi algorithm gave us approximately 87% accuracy. The 13% loss of accuracy, to a large part, can be attributed to the fact that when the algorithm hits an unknown word (i.e. not present in the training set), it naively assigns the first tag in the list of tags that we have created. In this case, the first tag happens to be JJ, so it assigns JJ to all unknown words.

### Improving the Performance of the POS Tagger

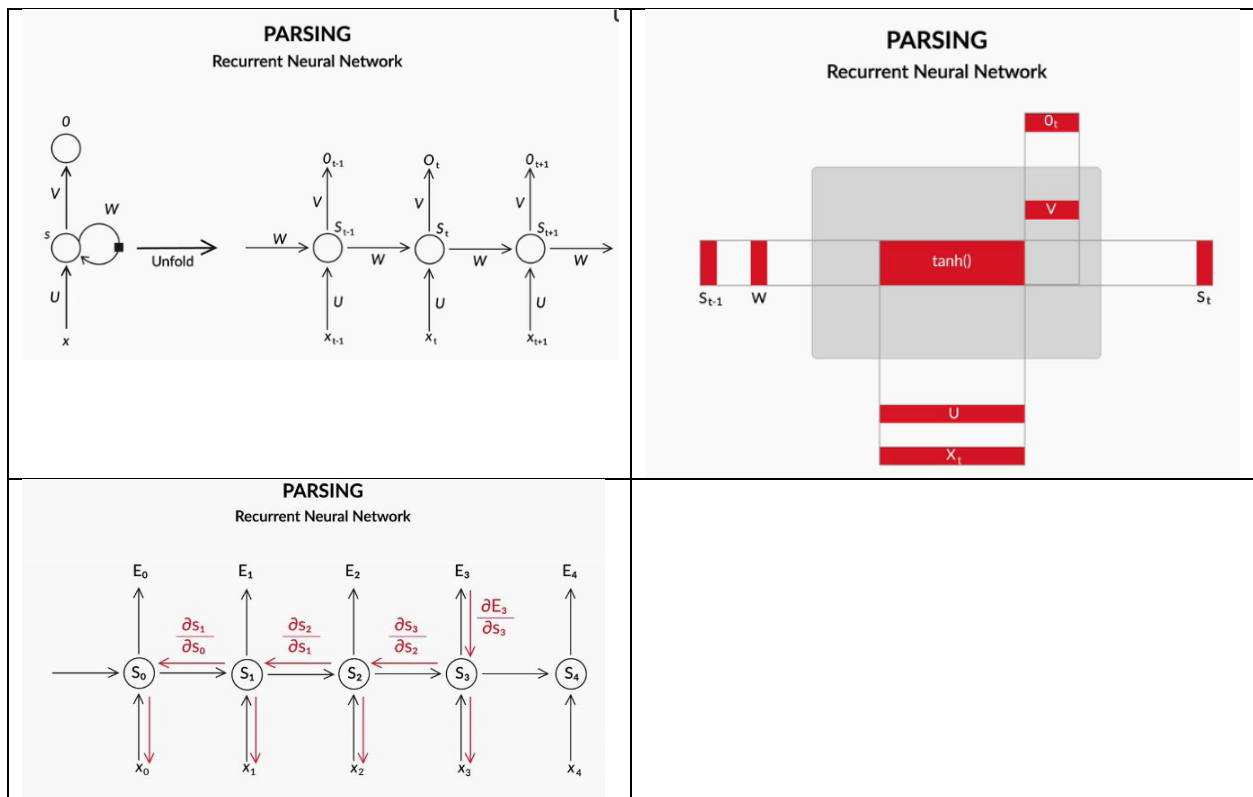
Improve the accuracy of the model further by making some modifications. Start by tagging the unknown words more smartly. Also, using some exploratory analysis, identify whether any particular sets of tags or words constitute to a large number of errors, understand the source of those errors, and address them head-on. These are left as exercises for you.

Next segment covers the fundamentals of **Recurrent Neural Networks (RNNs)** and how they are used for POS tagging.

## Deep Learning Based POS Taggers

Apart from conventional sequence models such as HMMs, significant advancement has been made in the field of Neural Networks (or deep learning) based sequence models. Specifically, **Recurrent Neural Networks (RNNs)** have empirically proven to outperform many conventional sequence models for tasks such as POS tagging, entity recognition, dependency parsing etc.

RNNs is in detail later in the Neural Network course. This segment gives a brief overview of how to build POS taggers using RNNs.



## Summary

Fundamentals of syntactic parsing and POS tagging.

There are three broad levels of syntactic processing –

- POS tagging
- Constituency parsing
- Dependency parsing

POS tagging is a crucial task in syntactic processing and is used as a **pre-processing step** in many NLP applications.

Various approaches to POS tagging:

- Lexicon-based
- Rule-based
- Stochastic models such as HMMs
- Deep-learning, RNN-based models

**Markov processes** and **HMMs** for POS tagging. A Markov process is used to model sequential phenomena and that the **first-order Markov assumption** states that the probability of an event (or state) depends only on the previous state.

The **Hidden Markov Model** is used to model phenomena where the **states are hidden**, and they **emit observations**. The **transition and the emission probabilities** specify the probabilities of transition between states and emission of observations from states, respectively. In POS tagging, the states are the POS tags while the words are the observations.

After **learning the HMM model parameters**, i.e. computing the transition and the emission probabilities, the **Viterbi algorithm** is used to assign POS tags efficiently, which would otherwise have been a computationally expensive problem. The Viterbi algorithm, although greedy in nature, is a **computationally viable** alternative to choosing a tag sequence from a large set of possible sequences.

Build your own HMM-based POS tagger and implement the Viterbi algorithm using the Penn Treebank training corpus.

Next session on **constituency parsing** will move a step higher and use POS tags to parse sentences.





