upGrad

Summary

Introduction to Syntactic Processing and PoS Tagging

You will be introduced to the following topics in this session:

- Syntax and syntactic processing
- Parts of speech and PoS tagging
- PoS tagging techniques: Rule-based and Hidden Markov Model
- PoS tagging code demonstration

Syntax and Syntactic Processing

Welcome to the first segment of this module on syntactic processing. We all have learnt in our schools how a particular language is built based on the grammatical rules.

Let's take a look at the sentences given below.

'Is EdTech upGrad company an.'

'EdTech is an company upGrad.'

'upGrad is an EdTech company.'

All of these sentences have the same set of words, but only the third one is syntactically or grammatically correct and comprehensible.

One of the most important things that you need to understand here is that in lexical processing, all of the three sentences provided above are similar to analyse because all of them contain exactly the same tokens, and when you perform lexical processing steps such as stop words removal, stemming, lemmatization and TFIDF or countvectorizer creation, you get the same result for all of the three sentences. The basic lexical processing techniques would not be able to identify the difference between the three sentences. Therefore, more sophisticated syntactic processing techniques are required to understand the relationship between individual words in a sentence.

Arrangement of words in a sentence plays a crucial role in better understanding the meaning of the sentence. These arrangements are governed by a set of rules that we refer to as 'syntax', and the process by which a machine understands the syntax is referred to as syntactic processing.

Syntax: A set of rules that govern the arrangement of words and phrases to form a meaningful and well-formed sentence

Syntactic processing: A subset of NLP that deals with the syntax of the language

You also looked at some examples such as chatbots, speech recognition systems and grammar checking software that use syntactic processing to analyse and understand the meaning of the text.

Now, let's consider the following example.

'In Bangalore, it is better to have your own transport.'

'We use car pooling to transport ourselves from home to office and back.'

What do you think about the word 'transport' in these two sentences?

Parts of Speech

Let's start with the first level of syntactic analysis: **PoS (Parts of Speech) tagging**. To understand PoS tagging, you first need to understand what parts of speech are.

Let's consider a simple example given below.

'You are learning NLP at upGrad.'

From your knowledge of the English language, you are well aware that in this sentence, 'NLP' and 'upGrad' are nouns, the word 'learning' is a verb and 'You' is a pronoun. These are called the parts of speech tags of the respective words in the sentence. A word can be tagged as a noun, verb, adjective, adverb, preposition, etc., depending upon its role in the sentence. These tags are called the PoS tags.

Assigning the correct tag, such as noun, verb and adjective, is one of the most fundamental tasks in syntactic analysis.

Suppose you ask your smart home device the following question.

'Ok Google, where can I get the permit to work in Australia?'

The word 'permit' can potentially have two PoS tags: noun or 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'.

upGrad

Parts of speech (PoS) are the groups or classes of words that have similar grammatical properties and play similar roles in a sentence. They are defined based on how they relate to the neighbouring words.

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.

Now, let's understand the types of PoS tags available in the English language.

A PoS tag can be classified in two ways: open class and closed class.
Open class refers to tags that are evolving over time and where new words are being added for the PoS tag.

- Open class
  - Noun
  - Verb
  - Adjective
  - Adverb
  - Interjection

Some useful examples of open class PoS tags are as follows:

- Name of the person
- Words that can be added or taken from another language such as words taken from the Sanskrit language such as 'Yoga' or 'Karma'
- Words that are nouns but can be used as verbs such as 'Google'
- Words that are formed by a combination of two words such as football, full moon and washing machine

Closed class refers to tags that are fixed and do not change with time.

- Closed class
  - Prepositions
  - Pronouns
  - Conjunctions
  - Articles
  - Determiners
  - Numerals

Some examples of closed class PoS tags are as follows:

- Articles: a, an, the
- Pronouns: you and I

You can take a look at the universal tagsets used by the SpaCy toolkit here.

Note: Spacy is an open-source library used for advanced natural language processing, similar to NLTK, which you have used in lexical processing.

In this module on syntactic processing, you are going to use the SpaCy library to perform syntactical analysis.

You do not need to remember all the PoS tags. You will pick up most of these tags as you work on the problems in the upcoming segments, but it is important to be aware of all the types of tags.

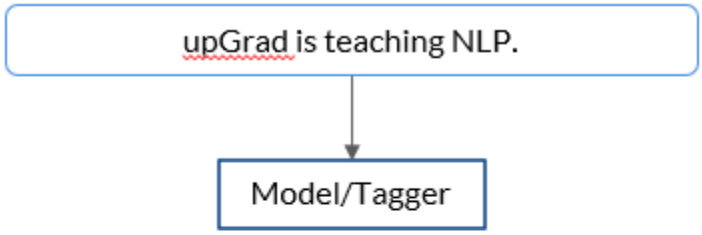
You can also refer to the alphabetical list of 36 part-of-speech tags used in the Penn Treebank Project, which is being used by the SpaCy library.

PoS Tagging

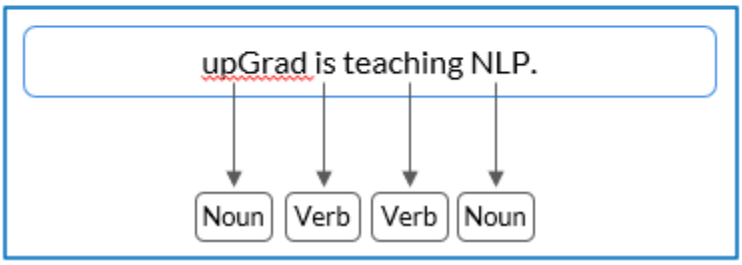
As of now, you have understood what the different part-of-speech tags are and their relevance in understanding the text. Now, let's understand how a machine assigns these PoS tags to the words in a sentence.

A PoS tagger is a model/algorithm that automatically assigns a PoS tag to each word of a sentence.

- Input:



- Output:



In this example, the input is "upGrad is teaching NLP.". When you put this sentence into a model or tagger, it will result in the output with respective PoS tags assigned to each word of the sentence such as 'upGrad' (noun), 'is' (verb), 'teaching' (verb) and 'NLP' (noun).

Let's take a look at another example given below.

Tagger input: 'She sells seashells on the seashore.'

Tagger output: 'She(PRON) sells (VERB) seashells(NOUN) on(SCONJ) the(DET) seashore(NOUN).'

You can refer to the universal PoS tag list for finding tags corresponding to their parts of speech and also refer to the alphabetical list of part-of-speech tags used in the Penn Treebank Project, which is being used by the SpaCy library.

Now, let's try to understand how one can build a simple rule-based PoS tagger to assign PoS tags to individual words in a sentence.

Suppose you have been given a training dataset that comprises words and their respective PoS tags' count. This is visually demonstrated in tabular format below.

Naive Model →

	Noun	Verb	Adjective	Adverb	Article	Pronoun
Word_1	2	0	6	3	0	0
Word_2	0	10	4	0	0	0
Word_3	0	0	0	0	0	34
Word_4	15	12	0	10	0	0
Word_5	0	23	4	16	0	0
Word_6	0	0	0	0	10	0

In this table, the word 'Word_1' occurs as a noun two times, and as an adjective, it occurs six times and so on in the training dataset.

The identification of PoS tags in the training dataset is done manually to predict the PoS tags of the test data.

In the table provided above, 'Word_1' appears as a noun two times, and as an adjective, it appears three times and so on. Now, suppose, you are given the following sentence (S).

S: "Word_4 Word_1 Word_3."

What will be the PoS tags of each word in this sentence?

you got the answer to the previous exercise, i.e., the PoS tags of the words of the sentence S will be as follows:

Word_4: Noun
Word_1: Adjective
Word_3: Pronoun

You assign the most frequent PoS tags that appear in the training data to the test dataset, and you realise that rule based tagger gives good results most of the time.

But, sometimes, it does not give satisfactory results because it does not incorporate the **context** of the word.

Let's take the following example.

Consider the word 'race' in both the sentences given below:

1. ‘The tortoise won the race.’
2. ‘Due to the injury, the horse will not be able to race today.’

In the first sentence, the word ‘race’ is used as a noun, but, in the second sentence, it is used as a verb. However, using the simple frequency-based PoS tagger, you cannot distinguish its PoS tags in different situations.

PoS Tagging: Hidden Markov Model - I

Let’s consider the following example; what does your mind process when you see the blank space at the end of this sentence?

Rahul is driving to _____.

Don’t you think that the blank space should be the name of a place?

How do you manage to identify that the blank space would be a name of the place?

Try to analyse your thoughts after reading this statement, when you see the word ‘Rahul’ who is driving to some place and hence, you reach to a conclusion that the blank space should be the name of a place (noun).

This means that you have sequentially looked at the entire sentence and concluded that the blank space should be the name of a place.

Sequence labelling is the task of assigning the respective PoS tags of the words in the sentence using the PoS tag of the previous word in the sentence.

Hidden Markov Model can be used to do sequence labelling, which means that it takes input of words in a sequence and assigns the PoS tags to each word based on the PoS tag of the previous word.

In the example ‘The tortoise won the race’, to decide the PoS tag of ‘race’, the model uses the PoS tag of the previous word, i.e., ‘the’, which is an article.

Now, let’s take a look at the following points and try to understand why Hidden Markov Model is called ‘Hidden’:

- When you observe (read/listen) a sentence, you only observe the words in the sentence and not their PoS tags; thus, PoS tags are hidden.
- You must infer these hidden tags from your observations, and that's why the Hidden Markov Model is called Hidden.

There are majorly two assumptions that HMM follows, which are as follows:

- The PoS tag of the next word is dependent only on the PoS tag of the current word.
- The probability of the next word depends on the PoS tag of the next word.

upGrad

Before learning about HMM, you need to understand the two most important types of matrices, i.e., emission and transition matrices.

To build any machine learning model, you first need to train that model using some training data, and then, you need to use that model to predict the output on the test data.

Here, the train data is the corpus of sentences, and you need to perform some manual tasks to assign the PoS tags to each word in the corpus. Once you manually assign the PoS tags to each word in the training corpus, you create two important matrices using the training dataset, which are as follows:

1. Emission matrix
2. Transition matrix

Note: We have used only two PoS tags, i.e., noun and verb, as of now to understand the concept in a simpler manner.

Emission matrix: This matrix contains all words of the corpus as row labels; the PoS tag is a column header, and the values are the conditional probability values.

Note: Conditional probability is defined as the probability of occurrence of one event given that some other event has already happened. You will get more idea about this in the following example.

Emission Matrix

	Noun	Verb
judge	0.18	0.16
praise	0.04	0.04
place	0.15	0.69
laugh	0.06	0.84
traffic	0.20	0.009
election	0.23	0.009
jury	0.2	0.009
SUM	**1**	**1**

Next Whenever a noun appears in the training corpus, there is an 18% chance that it will be the word ‘judge’. Similarly, whenever a verb appears in the training corpus, there is an 84% chance that it will be the word ‘laugh’.

So, here, 0.18 is the probability of occurrence of the word ‘judge’ given that there will be a noun at that place. In a similar way, 0.16 is the probability of occurrence of the word ‘judge’ given that there will be a verb at that place.

Transition matrix: This matrix contains PoS tags in the column and row headers. Let’s try to understand the conditional probability values that have been given in the following table.

Let's take a look at the first row of the table; it represents that 0.26 is the probability of occurrence of a noun at the start of the sentence in the training dataset. In the same way, 0.73 is the probability of occurrence of a verb at the start of the sentence in the training dataset.

If you take a look at the second row, then you will see that 0.44 is the probability of occurrence of aoun just after a noun in the training dataset, and similarly, 0.19 is the probability of occurrence of a verb just after a noun in the training dataset.

Transition Matrix

	Noun	Verb	<End>	SUM
<Start>	0.26	0.73	0	1
Noun	0.44	0.19	0.36	1
Verb	0.36	0.23	0.41	1
.	.	.	.	.
.	.	.	.	.

Essentially, the transition matrix gives the information of the next PoS tag considering the current PoS tag of the word.

Hidden Markov Model - II

In HMM, you understood that **transition and emission matrices** specify the probabilities of transition between tags and the emission of the word from the tag, respectively.

In this segment, you will learn how, with the use of transition and emission matrices, you can assign the correct PoS tag sequence based on the probability values.

Please note that you will not be gaining an in-depth understanding of HMM in this module, as it is not industrially relevant at all. You will only gain an intuitive understanding of how HMM works on sequence labelling of PoS tags.

Suppose you have the following corpus of the training dataset.

Sumit
Amit
Reena
Emily
Teaches
Writes
Learns
Reads
NLP
ML

The emission and transition matrices of this corpus are as follows:

Emission Matrix

	Noun	Verb
Sumit	0.2	0.025
Amit	0.1	0.025
Reena	0.1	0.025
Emily	0.2	0.025
Teaches	0.05	0.3
Writes	0.05	0.2
Learns	0.05	0.1
Reads	0.05	0.25
NLP	0.1	0.025
ML	0.1	0.025

Transition Matrix

	Noun	Verb	<End>
<Start>	0.7	0.3	0
Noun	0.2	0.5	0.3
Verb	0.65	0.1	0.25

Suppose you have been given the following test sentence to predict the correct PoS tags of the words.

S: “Sumit teaches NLP.”

As of now, we are only considering the two PoS tags, i.e., noun (N) and verb (V).

There are many combinations of PoS tags possible for the sentence ‘S’ such as NVN, NNN, VVV and VVN or NNV.

If you calculate the total number of combinations for this example, then you will see that there are only noun and verb tags; hence, 2^3 will be the total number of combinations possible.

Let’s consider the two sequences as of now, which are NNN and NVN.

Calculate the score for the NNN sequence.

Score of NNN:
[P(Start-Noun) * P(Sumit|Noun)] * [P(Noun-Noun)*P(teaches|Noun)] * [P(Noun-Noun) * P(NLP|Noun)]

(0.7*0.2) * (0.2*0.05) * (0.2*0.1) = 0.000028

Score of NVN:
[P(Start-Noun)*P(Sumit|Noun)] * [P(Noun-Verb)*P(teaches|Verb)] * [P(Verb-Noun) * P(NLP|Noun)]

(0.7*0.2) * (0.5*0.3) * (0.65*0.1) = 0.001365

You get the maximum score for the NVN sequence; hence the model assigns the noun to 'Sumit', the verb to 'teaches' and another noun to 'NLP' in the test sentence.

In practice, you do not have to calculate these scores from scratch. You will be using an already prepared toolkit to implement HMM, but it is important to be aware of the basic techniques used in PoS tagging.

Although you are not going to implement these PoS tagging techniques in the real-life scenario, you are going to use the SpaCy library to tag the correct PoS tags that are based on the neural network models. It is important to have an intuitive understanding of these techniques, including the rule-based tagger and HMM, to understand how a PoS tagger works.

With this, you have understood the theory part of PoS tagging.

PoS Tagging Application Part- I

You learnt about the underlying process of PoS tagging, i.e Hidden Markov Model(HMM). In this segment, you have gone through a coding exercise to see how PoS tagging can be used to solve the problem.

We are not going to design and implement the rule-based tagger or a sequential tagger like HMM in the real world. However, it is good to have an intuitive understanding of how a PoS tagger can be built.

For the implementation of PoS tagging, you are going to use a Python library, i.e., Spacy.

Please note that it is not important to deep dive into the working of the Spacy library. The working of Spacy is based on neural networks which is out of the scope of this course. Spacy works well in different applications of NLP such as PoS tagging and lexical processing steps. You just need to learn how to use this library for various applications.

Now, let's learn how one can use PoS tagging in applications like Heteronyms identification using the Spacy library.

Consider the following two sentences and pay attention to the pronunciation of word **'wind'** while you read:

- The doctor started to wind the bandage around my finger.
- The strong wind knocked down the tree.

You can listen to these sentences in the Google Translator application.

As you might have noticed, the pronunciation of **'wind'** is different in both the sentences, though its spelling is the same. Such words are known as **Heteronyms.**

Heteronyms are words that have the same spelling but mean differently when pronounced differently.

But how do machines like Alexa and Google Translator detect these heteronyms?

Let's see the following sentence:

'She wished she could desert him in the desert.'

Google translator pronounces 'desert' differently based on its PoS tag. You can see that both the instances of the word 'desert' have different PoS tags, one is a verb and other one is a noun.

PoS Tagging Application Part - II

Let's try to understand each line of code one by one.

model = spacy.load("en_core_web_sm")

- **'en'** stands for English language, which means you are working specifically on English language using the Spacy library.
- **'core'** stands for core NLP tasks such as lemmatization or PoS tagging, which means you are loading the pre-built models which can perform some of the core NLP-related tasks.
- **'web'** is the pre-built model of the Spacy library which you will use for NLP tasks that are trained from web source content such as blogs, social media and comments.
- **'sm'** means small models which are faster and use smaller pipelines but are comparatively less accurate. As a complement to 'sm', you can use **'lg'** or **'md'** for larger pipelines which will be more accurate than 'sm'.

Visit this link to know more about the Spacy package.

So, you have loaded the 'en_core_web_sm' model into the object 'model'. Now, let's apply the model on the input sentence, i.e., 'She wished she could desert him in the desert', using the following line of code.

#Use the model to process the input sentence

tokens = model("She wished she could desert him in the desert.")

So, you get the words and its part of speech and tags in the variable ‘tokens’. Now, to print the tokens, part of speech and PoS tags, you need to apply a ‘for’ loop on this ‘tokens’ object as follows:

```
# Print the tokens and their respective PoS tags.
for token in tokens:
    print(token.text, "--", token.pos_, "--", token.tag_)
```

So, when you print ‘token.pos_’, it prints the part of speech and using ‘token.tag_’, it prints the PoS tags corresponding to each token in a given sentence.

Please note that the ‘token.pos_’ gives you the part of speech which is defined in Spacy’s universal part of speech tags that you can get from this link. Moreover, to give the PoS tags using ‘token.tag_’, Spacy uses the PoS tags provided by the Penn treebank that you can get from this link.

When you get the PoS tags of each token of the sentence ‘She wished she could desert him in the desert’, then you will observe that the PoS tag of the word ‘desert’ is different in both of the instances. At one place, it is working as a **verb** and in another instance, it is working as a **noun.**

Word sense disambiguation (WSD): WSD is an open problem in computational linguistics concerned with **which sense of word** is used in a sentence. It is very difficult to fully solve the WSD problem. Google, however, has partially solved it.

Let’s try to listen to the pronunciation of the word ‘bass’ in the following sentence in Google Translator.

‘The bass swam around the bass drum on the ocean floor’.

However, as you have seen, the use of PoS tagging in heteronyms detection can be one of the prominent solutions to remove ambiguity in the sentence.

Let’s take the example of the following sentence:

‘She wished she could desert (verb) him in the desert (noun)’.

Here, the word ‘desert’ has two PoS tags based on its uses. At one place, it is working as a verb and at another place, it is working as a noun.

But what if the heteronyms have the same PoS tag but different pronunciation?

PoS tagging works pretty accurately to detect heteronyms but fails to distinguish between the words having the same spelling and the same PoS tags. Let’s consider the following example:

‘The bass swam around the bass drum on the ocean floor’.

When you implement the model on this sentence, you get the list of PoS tags of each token in the sentence. As you noticed that the word ‘bass’ is working as a noun at both of the places, but it should have different pronunciation at both the instances.

So, the problem when the system is not able to identify the correct pronunciation of the words which have the same PoS tag but different meanings in different contexts can be considered under the WSD problem.

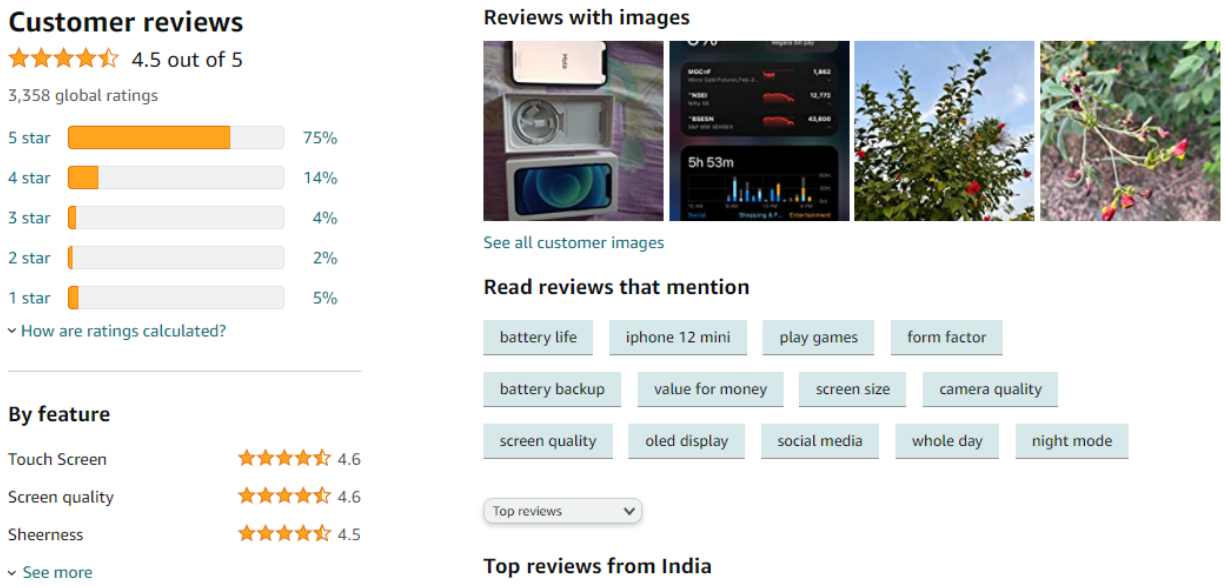
Please note that this is just one of the dimensions of WSD. WSD is altogether a broader area to discover and it is an open problem in computational linguistics concerned with identifying which sense of a word is used in a sentence.

PoS Tagging Case Study Part - I

Now, let’s come to a very interesting case study using PoS tagging. Here, we will provide you with the problem statement and some hints to produce output of each step involved to solve the case study.

Let’s understand the problem statement first.

You have used this feature of Amazon many times before buying any product from the website where you look into the reviews of the product category wise. Let’s look at the following.



The image is the Amazon page of Apple iPhone 12 Mini mobile phone. As you can see, there is categorisation of reviews into features based on the reviews’ text submitted by the users. These features can be ‘battery life’, ‘value for money’, ‘screen size’ and so on.

It is interesting to know that these features have been created to categorise the reviews after looking into the reviews' text given by multiple users. So, when you click on let's say 'battery life', you will get all the reviews related to battery life of this phone.

Please note that in this case study, we are not going to segregate or categorise the reviews, but we are going to find out the top product features that people are talking about in their reviews. The categorisation of reviews among the product features is just a task to assign each review to its category which can be done using simple coding steps once the main NLP steps are done.

Please note that as this use case is not a graded component and is designed to be solved as a case study, you have been provided with the data set and the well-commented codes to solve it. ***It is not at all mandatory to solve each of the problems; you can watch the videos directly.***

Let's first find out the PoS tags of each token in the following sentences and observe the PoS tags of the words 'screen', 'battery' and 'speaker'.

sent1 = "I loved the screen on this phone".
sent2 = "The battery life on this phone is great".
sent3 = "The speakers are pathetic".

You have an understanding of getting the PoS tags from the previous segments and can write the code to get the PoS tag of each token for these three sentences. Now, based on the output that you get after having the PoS tags,

You observed that the product features such as screen, battery and speaker have a PoS tag of a Noun.

An important thing to note here is that suppose we are able to find the frequency count of all the nouns in our data set, then by looking at top-n nouns, we can find out what product features people are talking about.

You have been given a Samsung phone reviews data set with file name 'Samsung.txt' file. It contains the reviews about the phone. This data set is a txt file, and each review is in a new line. In the notebook, you have been given the code to load the data set into the notebook.

In this task, you need to calculate the total number of reviews in the data set where each review is in a new line. Let's solve the next question based on the defined task.

In the next part, you need to check whether the hypothesis, i.e., the product features that we are trying to extract are acting as nouns in the real-life data set such as 'Samsung.txt'. But before that, let's first try to understand why lemmatization of words is needed before extracting the nouns from the data set.

In this task, you need to identify the right approach before extracting the nouns from the data set.

So, you can see that there are many nouns in the review text but all of them are not at all relevant to identify the features. So, what should be the approach to get relevant nouns from the review text?

Let's follow the approach that first calculates the PoS tags of each token and then converts them into their lemma form so that each word turns into its root form. Now, once you find the lemma form of each token, you need to count the frequency of each word which is acting as a noun.

You need to calculate the frequency of each noun after performing the lemmatization in the first review of the data set and arrange them in the descending order of their frequency of occurrence as a noun.

So, as you can see, when you restrict yourself to the top occurring noun, you can get the desired result. Now, let's calculate the frequency of occurrence of nouns for each token in the first 1,000 reviews of the data set.

Let's extract all the nouns from the reviews (first 1,000 reviews) and look at the top five most frequently lemmatised noun forms.

You learnt that the top frequently occurring noun PoS tags can work as product features.

Gunnvant has used the **'tqdm'** library to track the time of the processing of 1,000 reviews. You have seen that to process the first 1,000 reviews, it takes 17 seconds and, hence, to process the entire data set, it will take around 782 seconds which is approximately 13 minutes.

PoS Tagging Case Study Part - II

You need to extract all the nouns from all of the reviews and look at the top 10 most frequently lemmatised noun forms. You are already aware that it will take too much time to process the entire data set (approximately 13 minutes that we calculated in the previous segment). To reduce the processing time, you can use the following line of code when you load the Spacy model.

```
# shorten the pipeline loading
nlp=spacy.load('en_core_web_sm',disable=['parser','ner'])
```

However, for solving the next question, please use the aforementioned lines of code.

Gunnvant has used the following lines of code to reduce the processing time:

```
# shorten the pipeline loading
nlp=spacy.load('en_core_web_sm',disable=['parser','ner'])
```

When you load the Spacy using ‘en-core_web_sm’, the system finds everything like PoS tags, pre-built NER, lemmatised forms and so on. When you disable the ‘parser’ and ‘ner’, you are actually telling the system to not perform these tasks which eventually reduces the time to process the data set. As you can see, the time to process the entire data set was calculated as approximately 13 minutes earlier. However, in this case, the total time to process the entire data set is 4.6 minutes only.

Let’s see some pointers

- Now, we know that people mention battery, product, screen, etc. However, we still do not know in what context they mention these keywords.
- The most frequently used lemmatized forms of nouns inform us about the product features people are talking about in product reviews.
- In order to process the review data faster, Spacy allows us to use the idea of enabling parts of the model inference pipeline via the spacy.loads() command and the disable parameter.

Now, let’s come to a very interesting exercise. You have been able to identify the keywords or features such as ‘battery’, ‘screen’ and ‘phone’. Now, let’s try to see in what context these keywords occur using the following examples:

- Good battery life
- Big screen size
- Large screen protector
- Long battery lasts

In these examples, you can see that the keyword ‘battery’ occurs in the context of ‘good battery life’ or ‘screen’ occurs in the context of ‘big screen size’ and so on.

You have been given a sentence and need to identify the prefix and the suffix of the keyword ‘battery’. Using prefixes and suffixes, you can identify the context of keywords.

Now, let’s extract all suffixes and prefixes of the keyword ‘battery’ in the whole ‘Samsung.txt’ data set.

```
#define the regular expression pattern.
pattern = re.compile("\w+\sbattery\s\w+")

#apply the regular expression pattern for the keyword ‘battery’ to all the reviews of the data set using ‘findall’ function.
prefixes_suffixes = re.findall(pattern,samsung_reviews)
# Add prefixes and suffixes in separate arrays for the entire dataset corresponding to the keyword ‘battery’.
```

```
prefixes = []
suffixes = []
for p in prefixes_suffixes:
    l = p.split(" ")
    prefixes.append(l[0].lower())
    suffixes.append(l[-1].lower())
```

print the prefixes in the descending order of their count corresponding to the keyword 'battery'.

```
pd.Series(prefixes).value_counts().head(5)
```

print the suffixes in the descending order of their count corresponding to the keyword 'battery'.

```
pd.Series(suffixes).value_counts().head(5)
```

A. You need to separate the suffixes and prefixes of all the combinations where a keyword 'battery' is present and then print the count of each prefix corresponding to the keyword 'battery' in a descending order.

As you have seen in the aforementioned example, there are many prefixes and suffixes which are less relevant to combine with keywords 'battery' such as 'the battery' and 'that phone' because these words are mainly stop words and do not carry any relevant information.

So, it will be better if you remove the stop words from the 'prefixes' and the 'suffixes'.

You need to remove the stop words from the arrays of 'prefixes' and 'suffixes' for the keyword 'battery'.

Before attempting the next question, you need to run the code that has been asked in the previous two questions so that you get the arrays named 'prefixes' and suffixes' for the keyword 'battery'.

So, you have learnt to get the most appropriate combination of keywords and its prefixes and suffixes after eliminating the stop words.

upGrad

Summary

Name Entity Recognition

In this session, the following topics have been covered:

1. Name Entity Recognition (NER)
2. Python implementation of NER

Named Entity Recognition

As you already know, with the help of PoS tagging, you can determine the relationship between the words in a sentence. Now, the next step of understanding the text is Named Entity Recognition (NER).

Consider the following two representations of the given sentence:

'*John bought 300 shares of Apple in 2006*'

Representation1: $[John]_{NOUN}$ bought 300 shares of $[Apple]_{NOUN}$ in $[2006]_{NUMBER}$

Representation2: $[John]_{PERSON}$ bought 300 shares of $[Apple]_{ORGANISATION}$ in $[2006]_{DATE}$

As you can see, Representation1 has tagged the words 'John' and 'Apple' as nouns and '2006' as a number.

On the other hand, Representation 2 indicates the entity of the words like 'John' is tagged as 'PERSON' and 'Apple' as 'ORGANISATION', which provides information about the entities present in the sentence. This output can be achieved by using NER techniques. Essentially, with the help of NER, you will be able to find what the word is referring to like 'John' is a person and 'Apple' is an organisation.

NER techniques are applied in various fields such as search engine, chatbot and mainly in entity extraction in the long texts such as reviews, books, blogs and comments.

Named Entity Recognition (NER) enables you to easily identify the key elements in a piece of text, such as a person's name, locations, brands, monetary values, dates and so on.

Some example sentences with their named-entity recognition are as follows:

Note that GPE is short for geopolitical entity, ORG is short for organisation and PER is short for person.

1. S: 'Why is Australia burning?'
   NER: 'Why is Australia$_{[GPE]}$ burning?'

2. S: 'UK exits EU'
   NER: 'UK$_{[GPE]}$ exits EU$_{[ORG]}$'

3. S: 'Joe Biden intends to create easier immigration systems to dismantle Trump's legacy'
   NER: 'Joe Biden$_{[PER]}$ intends to create easier immigration systems to dismantle Trump's$_{[PER]}$ legacy'

4. S: 'First quarter GDP contracts by 23.9%'
   NER: 'First quarter$_{[DATE]}$ GDP contracts by 23.9%$_{[PERCENT]}$'

Some commonly used entity types are as follows:

PER: Name of a person (John, James, Sachin Tendulkar)
GPE: Geopolitical entity (Europe, India, China)
ORG: Organisation (WHO, upGrad, Google)
LOC: Location (River, forest, country name)

In this course, we will be using the Spacy toolkit for NER tasks. Spacy contains predefined entity types and pretrained models, which makes our task easier.

Named Entity Recognition How to?

You learnt about the concept of named entities. In this segment, you will learn how the NER system works.

Some important points mentioned as follows:

Noun PoS tags: Most entities are noun PoS tags. However, extracting noun PoS tags is not enough because in some cases, this technique provides ambiguous results.

Let's consider the following two sentences:

S1: 'Java is an Island in Indonesia.'

S2: 'Java is a programming language.'

PoS tagging only identifies 'Java' as a noun in both these sentences and fails to indicate that in the first case, 'Java' signifies a location and in the second case, it signifies a programming language.

A similar example can be **'Apple'.** PoS tagging fails to identify that 'Apple' could either be an organisation or a fruit.

Let's take a look at a simple rule-based NER tagger.

Simple rule-based NER tagger: This is another approach to building an NER system. It involves defining simple rules such as identification of faculty entities by searching 'PhD' in the prefix of a person's name.

However, such rules are not complete by themselves because they only work on selected use cases. There will always be some ambiguity in such rules.

Therefore, to overcome these two issues, **Machine Learning** techniques can be used in detecting named entities in text.

IOB Labelling

Machine learning can prove to be a helpful strategy in named entity recognition. So, before understanding how exactly ML is used in the NER system, you will learn about IOB labelling and sequence labelling related to the NER system.

IOB (inside-outside-beginning) labeling is one of many popular formats in which the training data for creating a custom NER is stored. IOB labels are manually generated.

This helps to identify entities that are made of a combination of words like 'Indian Institute of Technology', 'New York' and 'Mohandas KaramChand Gandhi'.

Suppose you want your system to read words such as 'Mohandas Karamchand Gandhi', 'American Express' and 'New Delhi' as single entities. For this, you need to identify each word of the entire name as the PER (person) entity type in the case of, say, 'Mohandas Karamchand Gandhi'. However, since there are three words in this name, you will need to differentiate them using IOB tags.

The IOB format tags each token in the sentence with one of the following three labels: I - inside (the entity), O - outside (the entity) and B - at the beginning (of entity). IOB labeling can be especially helpful when the entities contain multiple words.

So, in the case of 'Mohandas Karamchand Gandhi', the system will tag 'Mohandas' as B-PER, 'Karamchand' as I-PER and 'Gandhi' as I-PER. Also, the words outside the entity 'Mohandas Karamchand Gandhi' will be tagged as 'O'.

Consider the following example for IOB labeling:

Sentence: 'Donald Trump visit New Delhi on February 25, 2020 "

Donald	Trump	visit	New	Delhi	on	February	25	,	2020
B-Person	I-Person	O	B-GPE	I-GPE	O	B-Date	I-Date	I-Date	I-Date

In the example above, the first word of more than one-word entities starts with a **B label,** and the next words of that entity are labelled as **I,** and other words are labelled as **O.**

Note that you will not always find the IOB format only in all applications. You may encounter some other labeling methods as well. So, the type of labelling method to be used depends on the scenario. Let's take a look at an example of a healthcare data set where the labelling contains 'D', 'T', and 'O', which stand for disease, treatment and others, respectively.

S: 'In[O] the[O] initial[O] stage[O], Cancer[D] can[O] be[O] treated[O] using[O] Chemotherapy[T]'

NER: Python Demonstration

You learnt about named entity types, the benefits of NER tagging over PoS tagging and IOB tags.

In this segment, you will perform a NER demonstration in Python using the Spacy library.

To understand the query, a search engine first tries to find the named entity used in the query, and then corresponding to that entity, it displays the appropriate searches to the user. Named entity recognition is one of the aspects of the search engine mechanism.

You obtained PoS tags of the following sentence:

'Sumit$_{[PROPN]}$ is$_{[AUX]}$ an$_{[DET]}$ adjunct$_{[ADJ]}$ faculty$_{[NOUN]}$ at$_{[ADP]}$ UpGrad$_{[PROPN]}$.'

However, PoS tagging failed to distinguish between 'Sumit' and 'UpGrad'.

According to PoS tagging, both 'Sumit' and 'UpGrad' are proper nouns.

Like PoS tagging, you can also find the named entities present in a sentence using the following code:

import spacy # import spacy module

```
model = spacy.load("en_core_web_sm") #load pre-trained model
doc = "Any sentence"

processed_doc = model(doc); #process input and perform NLP tasks
for ent in processed_doc.ents:
  print(ent.text, " -- ", ent.start_char, " -- ", ent.end_char, " -- ", ent.label_)
```

After processing the model over 'doc', we got 'processed_doc', which contains an attribute called 'ents', which contains all the entities that identify with the Spacy NER system. So, the code above iterates over every token of the sentence and provides the corresponding entity associated with that token that was identified by the Spacy-trained model.

You might have also noticed that the entity types of 'Sumit' and Upgrad in the following two sentences are different

'Sumit is an adjunct faculty of Upgrad$_{[GPE]}$.'

'Dr. Sumit$_{[PERSON]}$ is an adjunct faculty of Upgrad$_{[ORG]}$'

In the second sentence, the model successfully finds both the entities, but in the first sentence, it fails to identify 'Sumit'. The reason for this could be that 'Sumit' is not present in the corpus on which the model was trained. However, adding 'Dr' as a prefix indicates the model that the next word is a person. Based on these observations, we can conclude that there are various situations where systems make errors depending on the application.

Let's now consider a practical application of NER systems.

Anonymisation of data and redacting personally-identifying information

In many scenarios, we would want to withhold sensitive and confidential information such as names of persons, dates and amounts.

Suppose you are asked to write a program that can anonymise people's names in many emails for confidential purposes.

For this task, we can use NER techniques to automatically identify PERSONS in the text and remove PERSON names from the text.

Note: we have taken the example of emails from the Enron email data set for illustration in this demo.

- Email source:
http://www.enron-mail.com/email/lay-k/elizabeth/Christmas_in_Aspen_4.html

- Complete Enron data:
http://www.enron-mail.com/

We first identified the person's name in the email using *'ent.label== 'PERSON'.* We then replaced the character of a person's name with '*' to make it anonymous. Now, based on this anonymisation example, answer the following questions.