**Syntactic Processing**

Let's start the session with an understanding of terms like syntactic processing and syntax. We all have learned in our schools how a particular language is built based on grammatical rules.

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

‘Is EdTech coursera company an.’

‘EdTech is an company coursera.’

**‘coursera** **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 the same tokens, and when you perform lexical processing steps such as stop words removal, stemming, lemmatization and TFIDF or count vectorizer 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.

**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.

**Parts of Speech (PoS) Tagging**

Let’s start with the first level of syntactic analysis: PoS (Parts of Speech) tagging.

Let’s consider a simple example given below.

‘You are learning NLP at coursera.’

From your knowledge of the English language, you are aware that in this sentence, ‘NLP’ and ‘coursera’ 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'.

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. All the subsequent parsing techniques (constituency parsing, dependency parsing, etc.) use the part-of-speech tags to parse the sentence.

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](https://universaldependencies.org/docs/u/pos/).

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

You do not need to remember all the PoS tags. You will pick up most of these tags as you work on the problems, 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](https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html), which is being used by the spaCy library.

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

A screenshot of a computer

Description automatically generated

In this example, the input is ‘Coursera 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 ‘Coursera (noun), ‘is’ (verb), ‘teaching’ (verb) and ‘NLP’ (noun).

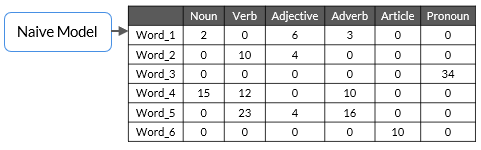
Let’s take a look at another example given below.

- **Tagger input**: ‘She sells seashells on the seashore.’

- **Tagger output**: ‘She(PRON) sells (VERB) seashells(NOUN) on(SCONJ) the(DET) seashore(NOUN).’

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.



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 adverb, 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?

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.

**Hidden Markov Model**

In the above segment, PoS tagging based on the frequency of tags alone, which seems inefficient when words are used in different contexts. So, to improve this model, in this segment, you will learn about the Hidden Markov Model, which performs better and uses the context of the previous word to decide the PoS tag of the current word.

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.

Now, what if we build an algorithm that can work sequentially to identify the PoS tags of the words based on the PoS tag of the previous word?

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.

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.

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 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.

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.

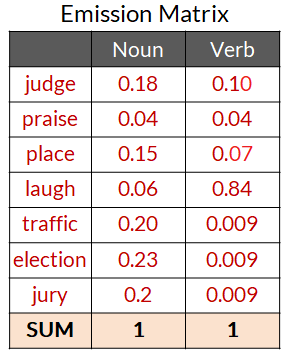
Here, the train data is the corpus of sentences, and you need to perform some manual tasks to assign the PoS tags to each word in the corpus. Once you manually assign the PoS tags to each word in the training corpus, you create two important matrices using the training dataset, which are as follows:

* Emission matrix
* 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.



P(judge|Noun) + P(praise|Noun) + … = 1

P(judge|Verb) + P(praise|Verb) + … = 1

For example, in the corpus that has been used in the video provided above, 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, 0.10 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.

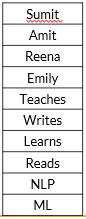
If you take a look at the second row, then you will see that 0.44 is the probability of occurrence of noun just after a noun in the training dataset, and similarly, 0.19 is the probability of occurrence of a verb just after a noun in the training dataset.

A table with numbers and symbols

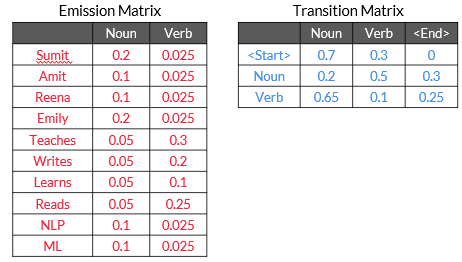
Description automatically generated

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

Suppose you have the following corpus of the training dataset.



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



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.

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](https://www.google.com/search?q=google+translate&rlz=1C1CHBF_enIN868IN868&oq=google+tra&aqs=chrome.0.0i131i433j0j69i57j0i131i433j0i433j69i60l3.5141j0j7&sourceid=chrome&ie=UTF-8) 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.

As you saw, Sumit took 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.

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’.

You will see in the upcoming videos that both the occurrences of the word ‘bass’ have the same PoS tags, but the pronunciation should be different. Even Google Translator is not able to pronounce it differently.

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.

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 in this sentence, you get the list of PoS tags of each token in the sentence. The word ‘bass’ is working as a noun at both of the places, but it should have different pronunciations in both instances.

So, the problem when the system is not able to identify the correct pronunciation of the words that 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.

**Constituency Parsing**

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

You need to learn techniques that can help you understand the grammatical structures of complex sentences. Constituency parsing and dependency parsing can help you achieve that.

Let’s understand parsing through an example. Suppose you ask a question answering (QA) system, such as Amazon's Alexa or Apple's Siri, the following question.

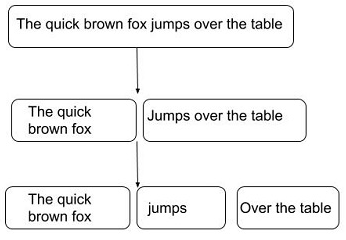
"Who won the FIFA World Cup in 2014?"

The QA system can respond meaningfully only if it understands that the phrase ‘FIFA World Cup' is related to the phrase 'in 2014'. The phrase 'in 2014' 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.

Let's take another example sentence to understand how a parsed sentence looks like.

"The quick brown fox jumps over the table."

The figure given below shows the three main constituents of this sentence.



This structure divides the sentence into the following three main constituents:

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

**Constituency parsing** is the process of identifying the constituents in a sentence and the relation between them.

For example, “Coursera is a great platform.”

Here, ‘Coursera is the noun phrase, and ‘is a great platform’ is the verb phrase.

The figure shown below represents the parse tree that shows how parsers implement parsing based on grammar.

Root

S

.

NP

VP

VBZ

NP

NN

.

NNP

Coursera

DT

a

JJ

is

platform

great

To summarise the chart given above, a constituent parse tree can be divided into three levels, which are as follows:

1. Sentence constituent:
   1. S (Coursera is a great platform)
   2. NP: Noun phrase (Coursera)
   3. VP: Verb phrase (is a great platform)
2. Sentence words: ‘Coursera, ‘is’, ‘a’, ‘great’, ‘platform’
3. Part-of-speech tags: NNP, VBZ, NP, DT, JJ, NN

A constituency parse tree always contains the words of a sentence as its terminal nodes. Usually, each word has a parent node containing its part-of-speech tag (noun, adjective, verb, etc.).

All the other non-terminal nodes represent the constituents of the sentence and are usually one of verb phrases, noun phrases or prepositional phrases (PP).

In this example, at the first level below the root, the sentence has been split into a noun phrase, made up of the single word “Coursera” and a verb phrase “is a great platform”. This means that grammar contains a rule such as S -> NP VP, meaning that a sentence can be created with the concatenation of a noun phrase and a verb phrase.

Similarly, the noun phrase is divided into a determiner, adjective and noun.

To summarise, constituency parsing creates trees containing a syntactical representation of a sentence according to a context-free grammar rule. This representation is highly hierarchical and divides the sentences into its single phrasal constituents.

‘**We saw the Statue of Liberty flying over New York.**’

Although having the same arrangement of words, the sentence can be interpreted in the following two ways:

1. Person saw that the ‘Statue of Liberty’ was flying.
2. A person is flying over New York he/she saw ‘Statue of Liberty’ from the top.

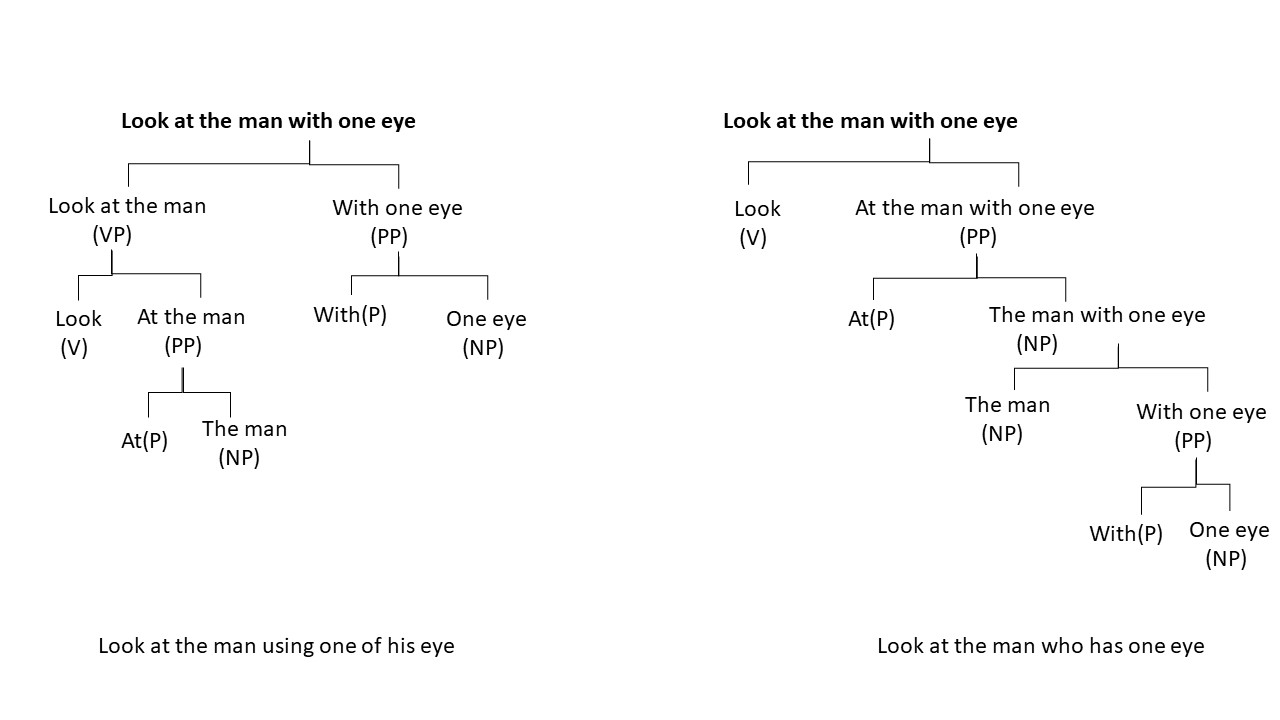
To understand ambiguity with a parse tree, let’s consider the following sentence.

“Look at the man with one eye.”

This sentence may have the following two meanings:

1. Look at the man using only one of his eyes.
2. Look at the man who has one eye.

Their respective parse trees are shown in the figure given below.



There are two parse trees possible for this sentence; if we are able to identify the relationship among the words instead of looking at individual constituents or PoS tags, then understanding each word with other words will be easier, and the machine will be able to understand the syntax or meaning of the sentence better. This relationship structure can be drawn using the dependency parsing technique.

In general, since natural languages are inherently ambiguous (at least for computers to understand), there are often cases where multiple parse trees are possible, such as those in the example provided above. So, to understand the relationship between different words, you need to use dependency parsing, which you will learn in the next segment.

**Dependency Parsing**

In dependency parsing, we do not form constituencies (such as noun phrase and verb phrase) but rather establish a dependency between the words themselves.

Dependency parsing identifies the relation between words and produces a concise representation of these dependencies.

In the example ‘Coursera is a great platform’, constituency parsing fails to explain how ‘great’ is related to the two nouns of the sentence ‘Coursera’ and ‘platform’. Is ‘great’ related more to ‘Coursera’ than to ‘platform’? Whereas dependency parsing tells us that ‘great’ is a modifier of ‘platform’. Similarly for the two nouns, ‘Coursera is the subject of another noun ‘platform’.

A black screen with white text

Description automatically generated

You can take a look at the universal dependency tags [here](https://universaldependencies.org/docs/en/dep/) that spaCy uses.

Now, let’s take another example of the dependency parse tree. It is as follows:

Sentence: “Economic news had little effect on financial markets.”

You can [visualize the dependency](https://explosion.ai/demos/displacy?text=Economic%20news%20had%20little%20effect%20on%20financial%20markets&model=en_core_web_sm&cpu=0&cph=0) parse of this sentence here. Also, in the diagram shown below, we have merged the phrases such as 'Economic news' and 'little effect'.

Let’s identify the role of each word one by one, starting with the root verb.

The word 'had' is the root.

The phrase ‘Economic news’ is the nominal subject (nsubj).

The phrase 'little effect' is the direct object (dobj) of the verb 'had'.

The word 'on' is a preposition associated with 'little effect'.

The noun phrase 'financial markets' is the object of 'on'.

Now, let’s take a look at the role of each word in the parse.

The word 'Economic' is the modifier of ‘news’.

news -> amod -> economic

The words ‘financial’ and ‘little’ modify the words 'markets' and 'effect', respectively.

effect -> amod -> little

markets -> amod -> financial

The two words ‘on’ and ‘markets’ have no incoming arcs. The word ‘on’ is dependent on the word ‘effect’ as a nominal modifier.

effect -> prep -> on

The word ‘markets’ is an object of the word ‘on’.

on -> pobj -> markets

Moving further, you need to keep the following points in mind while understanding the dependency parse model:

* Each word is a node in a parse tree.
* There is exactly one node with no incoming arc (root).
* Each non-root node has exactly one incoming arc.
* There is a unique path from the root node to each non-root node.

In this way, dependency parsers relate words to each other.

**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.

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 the geopolitical entity, ORG is short for organisation and PER is short for a person.

S: ‘Why is Australia burning?’

NER: ‘Why is Australia[GPE] burning?’

S: ‘UK exits EU’

NER: ‘UK[GPE] exits EU[ORG]’

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’

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)

**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.

**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 (inside-outside-beginning)** labelling 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 labelling 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 labelling:

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 | Idate |

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 labelling 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]’

**Conditional Random Field (CRF)**

Let’s take some examples:

* ‘I drove away in my Jaguar.’
* ‘The deer ran away seeing the Jaguar.’

The word ‘Jaguar’ in the first sentence refers to a car manufacturing company, and in the second sentence, it refers to a species of animal. Let’s first try to solve the question given below to find the NER tags or entity labels of the word ‘Jaguar’ in these two sentences.

Both the sentences above are assigned the same NER tag, i.e., ORG (organisation) for the word ‘Jaguar’ when we use the predefined NER model or off-the-shelf tools using Spacy.

So, we can conclude that you cannot perform Named Entity Recognition using a predefined model in Spacy all the time because you may get the wrong result, as shown in the previous example where you got the correct NER tag for the word ‘Jaguar’ when the word was used as a company’s name, but the word was not correctly tagged when the word ‘Jaguar’ was used as the name of a species of animal.

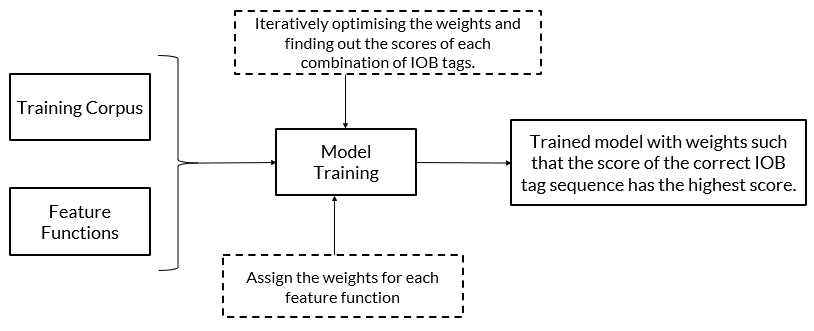
Can you recall the PoS tagging using the HMM model approach, which is based on the sequence labelling technique?

The same approach of sequence labelling can be used to perform Named Entity Recognition tasks.

The Conditional Random Field (CRF) can be used as a sequence labelling technique for performing NER tagging. CRF is used to perform custom NER.

Conditional Random Fields are the class of probabilistic models. There are two terms in the CRF nomenclature that you need to keep in mind, which are as follows:

* Random fields: These indicate that the CRF is probability a distribution-based machine learning model.
* Conditional: This indicates that the probabilities are conditional probabilities.



The above illustration is the overall architecture of the CRF model for custom NER.

The logic behind custom NER tagging using the CRF model can be divided into following parts:

1. Overview of custom NER using the CRF model
2. CRF model training
3. Model prediction

In most of the machine learning model building, we need to build the model using a training data set. Once the model is ready, we test the accuracy of the model or evaluate its performance using the test data set or, in other words, we perform the prediction using the test data set. Similarly, we will train the CRF model using the train data set and get the predictions using the already trained model.

In order to train or build the model, consider the following sentences as the training corpus:

X1= ‘Google Inc. is headquartered in Silicon Valley.’

X2= ‘I went to New York.’

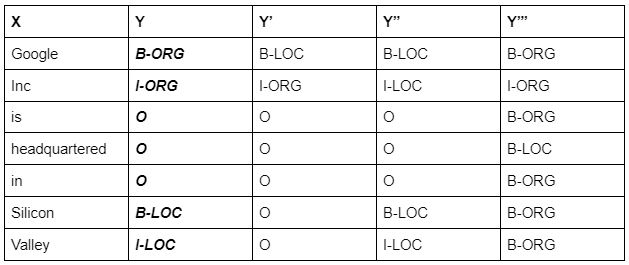
In the training data set, the NER tags such as IOB tags are assigned manually, as shown below.

* X1= Google (B-ORG) Inc (I-ORG) is (O) headquartered (O) in (O) Silicon (B-LOC) Valley (I-LOC).
* X2= I (O) went (O) to (O) New (B-LOC) York (I-LOC).

Where, ORG and LOC stand for organisation and location, respectively.

For a better understanding of the end-to-end process of training the model, we are considering only two IOB tags, ORG (organisation) and LOC (location).

To simplify, let’s consider only sentence ‘X1’ in the training corpus. For O, ORG and LOC, there can be multiple combinations of IOB tags possible for the model sentence ‘X1’. Some of them can be written as shown in the table below.



So, for the input sentence ‘X’, there can be multiple combinations of IOB tags possible, and some of them are shown in the above table. The highlighted one is the correct combination, which is tagged manually in the training data set. The model needs to train itself or assign the weights in such a way that it assigns the highest score to the correct combination.

Now, apart from the training data set, we require features and their values to train a model. Here, the training corpus is nothing but the text data. We need to create features out of this text data, which we can feed into the model.

In any model building task, we need to define the features we want to feed into the model. In custom NER applications, we can define the features using the CRF technique. After defining the features and obtaining their numerical values, we will understand how the model calculates weights and scores.

Some commonly used features in the CRF technique for NER applications are as follows:

* We can build logic on the input word ‘xi’ and on the surrounding words, which could be ‘xi-1’ or ‘xi+1’.
* The PoS tag of the word ‘xi’ and surrounding words.
* Is a particular word present in a dictionary (dictionary of common names, dictionary of organic chemicals, etc.)
* Word shapes:
  + 26-03-2021 => dd-dd-dddd
  + 26 Mar 2021 => dd Xxx dddd
  + W.H.O => X.X.X
* Presence of prefix and suffixes

As the first part of model building activity is to have a training data set. We have a training data set in which each token is manually tagged. The next part of the activity is to have some features to feed into the model. To get some features for this data set, it is necessary to define the feature functions for this data set. Below are some feature we can consider:

* f1 (X, xi, xi-1, i) = 1 if xi= Xx+; otherwise, 0 (Words starting with an uppercase letter)
* f2 (X, xi, xi-1, i) = 1 if xi= Noun and xi-1 is Noun; otherwise, 0 (Continuous entity)
* f3 (X, xi, xi-1, i) = 1 if xi = Inc and xi-1 = B-Org; otherwise, 0 (Company names often end with Inc.)

We can elaborate the definition of the above feature functions as follows:

* The f1 feature indicates that if a particular word in the given sentence starts with an uppercase letter, then assign 1 as the value of f1; otherwise, assign 0 as the value of f1 to this word.
* The f2 feature indicates that if a particular word in the given sentence has a PoS tag of noun and the word before it also has a PoS tag of noun, then assign 1 as the value of f2 as to this word; otherwise, assign 0 as the value of f2 to this word.
* The f3 feature indicates that if a particular word in the given sentence is ‘Inc’ and the word before it has the NER tag of B-ORG, then assign 1 the value of f3 to this word; otherwise, assign 0 as the value of f3 to this word.

**So, the model considers all the possible combinations of IOB tags of the given training example and calculates the scores of each combination using the weights corresponding to each feature function. The model starts the computation by taking any random initial weights for each feature function and iteratively modifies the weights until it reaches a stage where the score of the correct IOB tag sequence is the highest**.

Refer the [link](https://medium.com/data-science-in-your-pocket/named-entity-recognition-ner-using-conditional-random-fields-in-nlp-3660df22e95c#:~:text=CRF%20is%20amongst%20the%20most,denoted%20by%20y%E1%B5%A2%E2%82%8B%E2%82%81) for the working of CRF model.