Natural Processing Language

UNIT 1

By Shivani Deopa

Introduction to NLP:

What is NLP

- Natural language processing (NLP) is a field of computer science and a subfield of artificial intelligence that aims to make computers understand human language.
- NLP uses computational linguistics, which is the study of how language works, and various models based on statistics, machine learning, and deep learning.
- These technologies allow computers to analyze and process text or voice data, and to grasp their full meaning, including the speaker’s or writer’s intentions and emotions.
- NLP powers many applications that use language, such as text translation, voice recognition, text summarization, and chatbots.
- You may have used some of these applications yourself, such as voice-operated GPS systems, digital assistants, speech-to-text software, and customer service bots.
- NLP also helps businesses improve their efficiency, productivity, and performance by simplifying complex tasks that involve language.

NLP Techniques

- NLP encompasses a wide array of techniques that aimed at enabling computers to process and understand human language. These tasks can be categorized into several broad areas, each addressing different aspects of language processing. Here are some of the key NLP techniques:

1. **Text Processing and Preprocessing In NLP**

- Tokenization: Dividing text into smaller units, such as words or sentences.
- Stemming and Lemmatization: Reducing words to their base or root forms.
- Stopword Removal: Removing common words (like “and”, “the”, “is”) that may not carry significant meaning.
- Text Normalization: Standardizing text, including case normalization, removing punctuation, and correcting spelling errors.

2. **Syntax and Parsing In NLP**

- Part-of-Speech (POS) Tagging: Assigning parts of speech to each word in a sentence (e.g., noun, verb, adjective).
- Dependency Parsing: Analyzing the grammatical structure of a sentence to identify relationships between words.
- Constituency Parsing: Breaking down a sentence into its constituent parts or phrases (e.g., noun phrases, verb phrases).

3. **Semantic Analysis**

- Named Entity Recognition (NER): Identifying and classifying entities in text, such as names of people, organizations, locations, dates, etc.
- Word Sense Disambiguation (WSD): Determining which meaning of a word is used in a given context.
- Coreference Resolution: Identifying when different words refer to the same entity in a text (e.g., “he” refers to “John”).

4. **Information Extraction**

- Entity Extraction: Identifying specific entities and their relationships within the text.
- Relation Extraction: Identifying and categorizing the relationships between entities in a text.

5. **Text Classification in NLP**

- Sentiment Analysis: Determining the sentiment or emotional tone expressed in a text (e.g., positive, negative, neutral).
- Topic Modeling: Identifying topics or themes within a large collection of documents.
- Spam Detection: Classifying text as spam or not spam.

6. **Language Generation**

- Machine Translation: Translating text from one language to another.
- Text Summarization: Producing a concise summary of a larger text.
- Text Generation: Automatically generating coherent and contextually relevant text.

7. **Speech Processing**

- Speech Recognition: Converting spoken language into text.
- Text-to-Speech (TTS) Synthesis: Converting written text into spoken language.

8. **Question Answering**

- Retrieval-Based QA: Finding and returning the most relevant text passage in response to a query.
- Generative QA: Generating an answer based on the information available in a text corpus.

9. **Dialogue Systems**

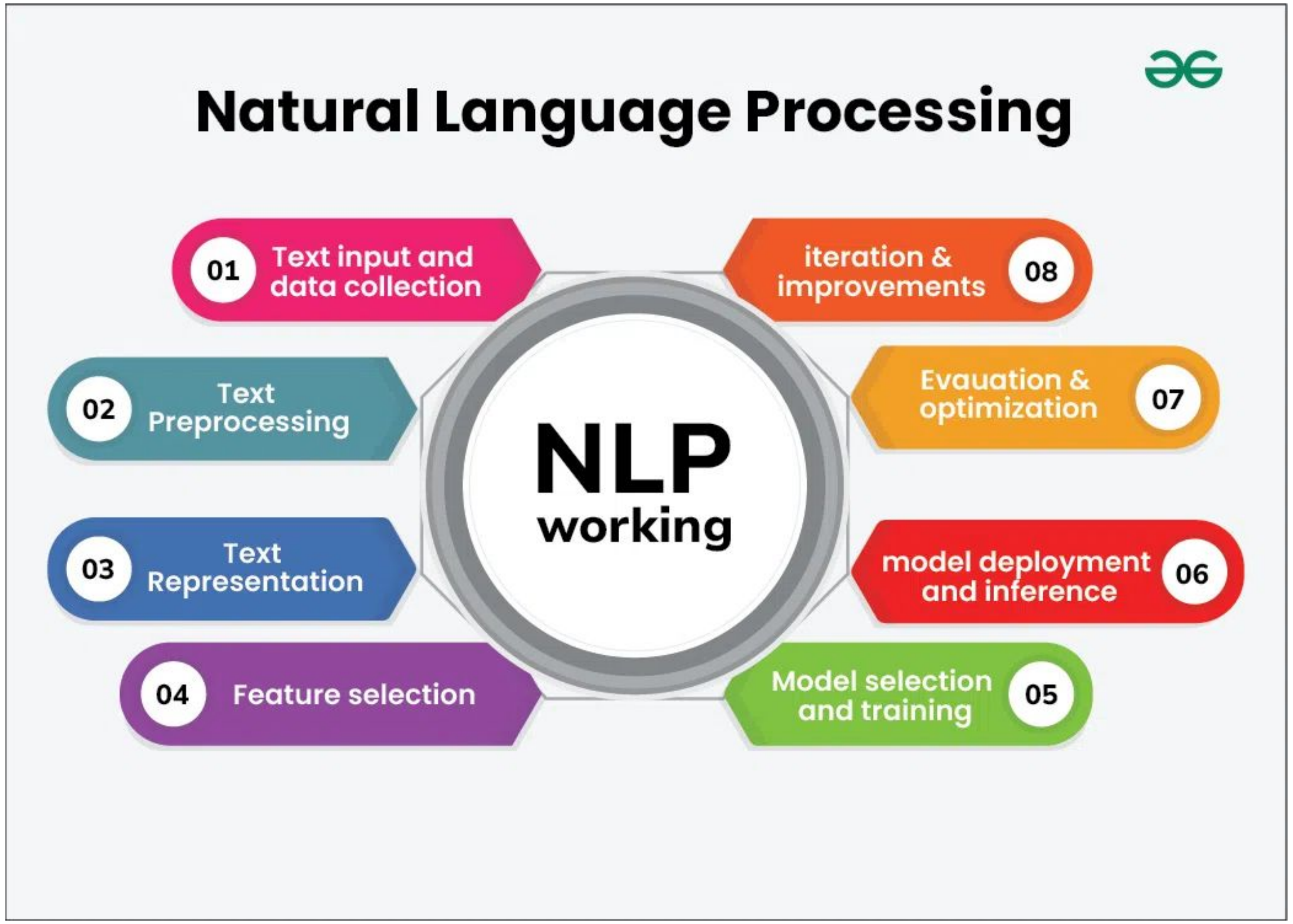
- Chatbots and Virtual Assistants: Enabling systems to engage in conversations with users, providing responses and performing tasks based on user input.

10. **Sentiment and Emotion Analysis in NLP**

- Emotion Detection: Identifying and categorizing emotions expressed in text.
- Opinion Mining: Analyzing opinions or reviews to understand public sentiment toward products, services, or topics.

Working of Natural Language Processing

- Working in natural language processing (NLP) typically involves using computational techniques to analyze and understand human language.
- This can include tasks such as language understanding, language generation, and language interaction.


Natural Language Processing
NLP working
01 Text input and data collection
02 Text Preprocessing
03 Text Representation
04 Feature selection
05 Model selection and training
06 model deployment and inference
07 Evauation & optimization
08 iteration & improvements

1. Text Input and Data Collection

- Data Collection: Gathering text data from various sources such as websites, books, social media, or proprietary databases.
- Data Storage: Storing the collected text data in a structured format, such as a database or a collection of documents.

2. Text Preprocessing

- Preprocessing is crucial to clean and prepare the raw text data for analysis. Common preprocessing steps include:
- Tokenization: Splitting text into smaller units like words or sentences.
- Lowercasing: Converting all text to lowercase to ensure uniformity.
- Stopword Removal: Removing common words that do not contribute significant meaning, such as “and,” “the,” “is.”
- Punctuation Removal: Removing punctuation marks.
- Stemming and Lemmatization: Reducing words to their base or root forms. Stemming cuts off suffixes, while lemmatization considers the context and converts words to their meaningful base form.
- Text Normalization: Standardizing text format, including correcting spelling errors, expanding contractions, and handling special characters.

3. Text Representation

- Bag of Words (BoW): Representing text as a collection of words, ignoring grammar and word order but keeping track of word frequency.
- Term Frequency-Inverse Document Frequency (TF-IDF): A statistic that reflects the importance of a word in a document relative to a collection of documents.
- Word Embeddings: Using dense vector representations of words where semantically similar words are closer together in the vector space (e.g., Word2Vec, GloVe).

4. Feature Extraction

- Extracting meaningful features from the text data that can be used for various NLP tasks.
- N-grams: Capturing sequences of N words to preserve some context and word order.
- Syntactic Features: Using parts of speech tags, syntactic dependencies, and parse trees.
- Semantic Features: Leveraging word embeddings and other representations to capture word meaning and context.

5. Model Selection and Training

- Selecting and training a machine learning or deep learning model to perform specific NLP tasks.
- Supervised Learning: Using labeled data to train models like Support Vector Machines (SVM), Random Forests, or deep learning models like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs).
- Unsupervised Learning: Applying techniques like clustering or topic modeling (e.g., Latent Dirichlet Allocation) on unlabeled data.
- Pre-trained Models: Utilizing pre-trained language models such as BERT, GPT, or transformer-based models that have been trained on large corpora.

6. Model Deployment and Inference

- Deploying the trained model and using it to make predictions or extract insights from new text data.
- Text Classification: Categorizing text into predefined classes (e.g., spam detection, sentiment analysis).
- Named Entity Recognition (NER): Identifying and classifying entities in the text.
- Machine Translation: Translating text from one language to another.
- Question Answering: Providing answers to questions based on the context provided by text data.

7. Evaluation and Optimization

- Evaluating the performance of the NLP algorithm using metrics such as accuracy, precision, recall, F1-score, and others.
- Hyperparameter Tuning: Adjusting model parameters to improve performance.
- Error Analysis: Analyzing errors to understand model weaknesses and improve robustness.

8. Iteration and Improvement

- Continuously improving the algorithm by incorporating new data, refining preprocessing techniques, experimenting with different models, and optimizing features.

Technologies related to Natural Language Processing

- There are a variety of technologies related to natural language processing (NLP) that are used to analyze and understand human language. Some of the most common include:

1. Machine learning: NLP relies heavily on machine learning techniques such as supervised and unsupervised learning, deep learning, and reinforcement learning to train models to understand and generate human language.
2. Natural Language Toolkits (NLTK) and other libraries: NLTK is a popular open-source library in Python that provides tools for NLP tasks such as tokenization, stemming, and part-of-speech tagging. Other popular libraries include spaCy, OpenNLP, and CoreNLP.
3. Parsers: Parsers are used to analyze the syntactic structure of sentences, such as dependency parsing and constituency parsing.

4. Text-to-Speech (TTS) and Speech-to-Text (STT) systems: TTS systems convert written text into spoken words, while STT systems convert spoken words into written text.
5. Named Entity Recognition (NER) systems: NER systems identify and extract named entities such as people, places, and organizations from the text.
6. Sentiment Analysis: A technique to understand the emotions or opinions expressed in a piece of text, by using various techniques like Lexicon-Based, Machine Learning-Based, and Deep Learning-based methods
7. Machine Translation: NLP is used for language translation from one language to another through a computer.
8. Chatbots: NLP is used for chatbots that communicate with other chatbots or humans through auditory or textual methods.
9. AI Software: NLP is used in question-answering software for knowledge representation, analytical reasoning as well as information retrieval.

Applications of NLP

1. Chatbots

- Chatbots are a form of artificial intelligence that are programmed to interact with humans in such a way that they sound like humans themselves.
- Depending on the complexity of the chatbots, they can either just respond to specific keywords or they can even hold full conversations that make it tough to distinguish them from humans.
- Chatbots are created using Natural Language Processing and Machine Learning, which means that they understand the complexities of the English language and find the actual meaning of the sentence and they also learn from their conversations with humans and become better with time.
- Chatbots work in two simple steps. First, they identify the meaning of the question asked and collect all the data from the user that may be required to answer the question. Then they answer the question appropriately.

2. Autocomplete in Search Engines

- Have you noticed that search engines tend to guess what you are typing and automatically complete your sentences? For example, On typing “game” in Google, you may get further suggestions for “game of thrones”, “game of life” or if you are interested in maths then “game theory”.
- All these suggestions are provided using autocomplete that uses Natural Language Processing to guess what you want to ask. Search engines use their enormous data sets to analyze what their customers are probably typing when they enter particular words and suggest the most common possibilities.
- They use Natural Language Processing to make sense of these words and how they are interconnected to form different sentences.

3. Voice Assistants

- These days voice assistants are all the rage! Whether its Siri, Alexa, or Google Assistant, almost everyone uses one of these to make calls, place reminders, schedule meetings, set alarms, surf the internet, etc. These voice assistants have made life much easier.
- But how do they work? They use a complex combination of speech recognition, natural language understanding, and natural language processing to understand what humans are saying and then act on it.
- The long term goal of voice assistants is to become a bridge between humans and the internet and provide all manner of services based on just voice interaction. However, they are still a little far from that goal seeing as Siri still can’t understand what you are saying sometimes!

4. Language Translator

- Want to translate a text from English to Hindi but don’t know Hindi? Well, Google Translate is the tool for you! While it’s not exactly 100% accurate, it is still a great tool to convert text from one language to another.
- Google Translate and other translation tools as well as use Sequence to sequence modeling that is a technique in Natural Language Processing. It allows the algorithm to convert a sequence of words from one language to another which is translation.
- Earlier, language translators used Statistical machine translation (SMT) which meant they analyzed millions of documents that were already translated from one language to another (English to Hindi in this case) and then looked for the common patterns and basic vocabulary of the language.
- However, this method was not that accurate as compared to Sequence to sequence modeling.

5. Sentiment Analysis

- Almost all the world is on social media these days! And companies can use sentiment analysis to understand how a particular type of user feels about a particular topic, product, etc.
- They can use natural language processing, computational linguistics, text analysis, etc. to understand the general sentiment of the users for their products and services and find out if the sentiment is good, bad, or neutral.
- Companies can use sentiment analysis in a lot of ways such as to find out the emotions of their target audience, to understand product reviews, to gauge their brand sentiment, etc.
- And not just private companies, even governments use sentiment analysis to find popular opinion and also catch out any threats to the security of the nation.

6. Grammar Checkers

- Grammar and spelling is a very important factor while writing professional reports for your superiors even assignments for your lecturers.
- After all, having major errors may get you fired or failed! That’s why grammar and spell checkers are a very important tool for any professional writer. They can not only correct grammar and check spellings but also suggest better synonyms and improve the overall readability of your content.
- And guess what, they utilize natural language processing to provide the best possible piece of writing! The NLP algorithm is trained on millions of sentences to understand the correct format.
- That is why it can suggest the correct verb tense, a better synonym, or a clearer sentence structure than what you have written. Some of the most popular grammar checkers that use NLP include Grammarly, WhiteSmoke, ProWritingAid, etc.

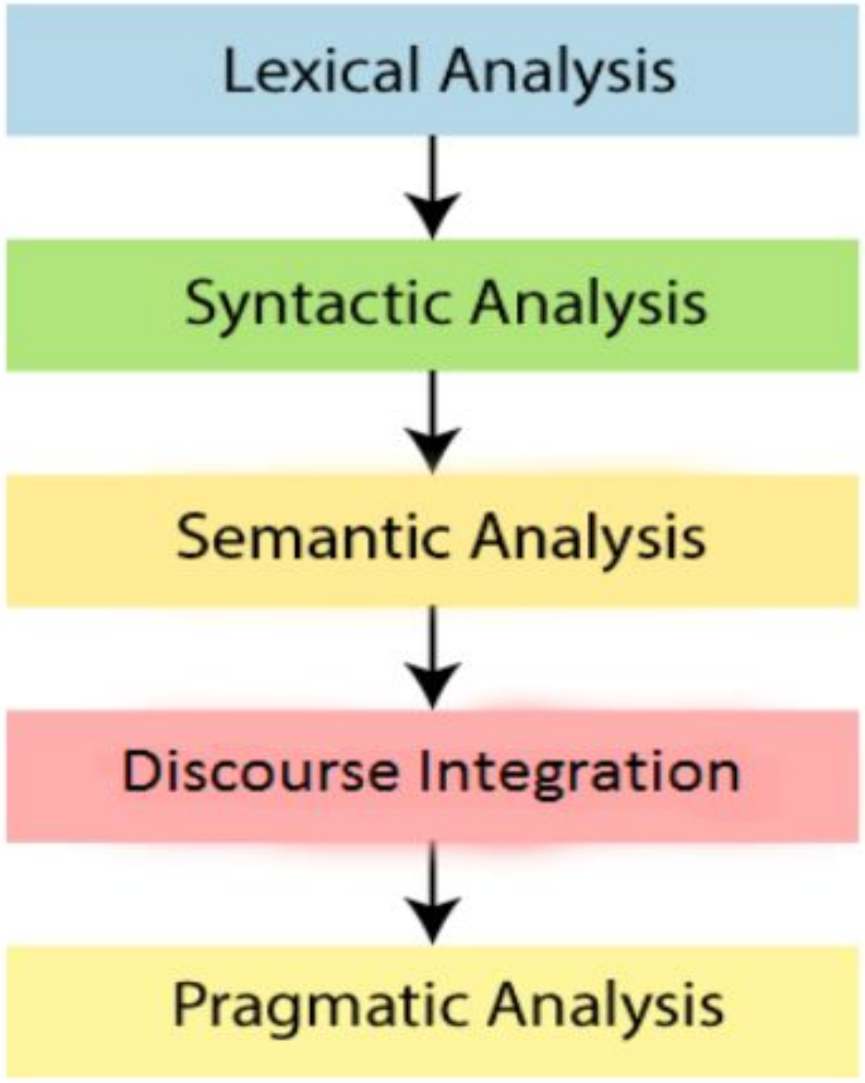
7. Email Classification and Filtering

- Emails are still the most important method for professional communication. However, all of us still get thousands of promotional Emails that we don’t want to read.
- Thankfully, our emails are automatically divided into 3 sections namely, Primary, Social, and Promotions which means we never have to open the Promotional section! But how does this work? Email services use natural language processing to identify the contents of each Email with text classification so that it can be put in the correct section. This method is not perfect since there are still some Promotional newsletters in Primary, but its better than nothing.
- In more advanced cases, some companies also use specialty anti-virus software with natural language processing to scan the Emails and see if there are any patterns and phrases that may indicate a phishing attempt on the employees.

NLP phases

- There are the following five phases of NLP:

1. Lexical Analysis
2. Syntactic Analysis
3. Semantic Analysis
4. Discourse Integration
5. Pragmatic Analysis



1. Lexical Analysis and Morphological

The first phase of NLP is the Lexical Analysis. This phase scans the source code as a stream of characters and converts it into meaningful lexemes. It divides the whole text into paragraphs, sentences, and words.

2. Syntactic Analysis (Parsing)

Syntactic Analysis is used to check grammar, word arrangements, and shows the relationship among the words.

Example: Agra goes to the Poonam

In the real world, Agra goes to the Poonam, does not make any sense, so this sentence is rejected by the Syntactic analyzer.

3. Semantic Analysis

Semantic analysis is concerned with the meaning representation. It mainly focuses on the literal meaning of words, phrases, and sentences.

4. Discourse Integration

Discourse Integration depends upon the sentences that proceeds it and also invokes the meaning of the sentences that follow it.

5. Pragmatic Analysis

Pragmatic is the fifth and last phase of NLP. It helps you to discover the intended effect by applying a set of rules that characterize cooperative dialogues.

For Example: "Open the door" is interpreted as a request instead of an order.

Difficulty of NLP including ambiguity

Natural Language Processing (NLP) faces various challenges due to the complexity and diversity of human language. Let’s discuss ***10 major challenges in NLP:***

1. Language differences

- The human language and understanding is rich and intricated and there many languages spoken by humans. Human language is diverse and thousand of human languages spoken around the world with having its own grammar, vocabular and cultural nuances.
- Human cannot understand all the languages and the productivity of human language is high. There is ambiguity in natural language since same words and phrases can have different meanings and different context. This is the major challenges in understating of natural language.
- There is a **complex syntactic** structures and grammatical rules of natural languages. The rules are such as word order, verb, conjugation, tense, aspect and agreement.
- There is rich semantic content in human language that allows speaker to convey a wide range of meaning through words and sentences.
- Natural Language is pragmatics which means that how language can be used in context to approach communication goals. The human language evolves time to time with the processes such as **lexical change.** The change in language represents cultural, social and historical factors.

2. Training Data

- Training data is a curated collection of input-output pairs, where the input represents the features or attributes of the data, and the output is the corresponding label or target. Training data is composed of both the features (inputs) and their corresponding labels (outputs).
- For NLP, features might include text data, and labels could be categories, sentiments, or any other relevant annotations.
- It helps the model generalize patterns from the training set to make predictions or classifications on new, previously unseen data.

3. Development Time and Resource Requirements

Development Time and Resource Requirements for ***Natural Language Processing (NLP)*** projects depends on various factors consisting the task complexity, size and quality of the data, availability of existing tools and libraries, and the team of expert involved. Here are some key points:

- **Complexity of the task:** Task such as classification of text or analyzing the sentiment of the text may require less time compared to more complex tasks such as machine translation or answering the questions.
- **Availability and Quality Data:** For Natural Language Processing models requires high-quality of annotated data. It can be time consuming to collect, annotate, and preprocess the large text datasets and can be resource-intensive specially for tasks that requires specialized domain knowledge or fine-tuned annotations.
- **Selection of algorithm and development of model:** It is difficult to choose the right algorithms machine learning algorithms that is best for Natural Language Processing task.
- **Evaluation and Training:** It requires powerful computation resources that consists of powerful hardware (GPUs or TPUs) and time for training the algorithms iteration. It is also important to evaluate the performance of the model with the help of suitable metrics and validation techniques for conforming the quality of the results.

4. Navigating Phrasing Ambiguities in NLP

It is a crucial aspect to navigate phrasing ambiguities because of the inherent complexity of human languages. The cause of phrasing ambiguities is when a phrase can be evaluated in multiple ways that leads to uncertainty in understanding the meaning. Here are some key points for navigating phrasing ambiguities in NLP:

- **Contextual Understanding:** Contextual information like previous sentences, topic focus, or conversational cues can give valuable clues for solving ambiguities.
- **Semantic Analysis:** The content of the semantic text is analyzed to find meaning based on word, lexical relationships and semantic roles. Tools such as word sense disambiguation, semantics role labeling can be helpful in solving phrasing ambiguities.
- **Syntactic Analysis:** The syntactic structure of the sentence is analyzed to find the possible evaluation based on grammatical relationships and syntactic patterns.
- **Pragmatic Analysis:** Pragmatic factors such as intentions of speaker, implicatures to infer meaning of a phrase. This analysis consists of understanding the pragmatic context.
- **Statistical methods:** Statistical methods and machine learning models are used to learn patterns from data and make predictions about the input phrase.

5. Misspellings and Grammatical Errors

Overcoming Misspelling and Grammatical Error are the basic challenges in NLP, as there are different forms of linguistics noise that can impact accuracy of understanding and analysis. Here are some key points for solving misspelling and grammatical error in NLP:

- **Spell Checking:** Implement spell-check algorithms and dictionaries to find and correct misspelled words.
- **Text Normalization:** The is normalized by converting into a standard format which may contains tasks such as conversion of text to lowercase, removal of punctuation and special characters, and expanding contractions.
- **Tokenization:** The text is split into individual tokens with the help of tokenization techniques. This technique allows to identify and isolate misspelled words and grammatical error that makes it easy to correct the phrase.
- **Language Models:** With the help of language models that is trained on large corpus of data to predict the likelihood of word or phrase that is correct or not based on its context.

6. Mitigating Innate Biases in NLP Algorithms

It is a crucial step of mitigating innate biases in NLP algorithm for conforming fairness, equity, and inclusivity in natural language processing applications. Here are some key points for mitigating biases in NLP algorithms.

- **Collection of data and annotation:** It is very important to confirm that the training data used to develop NLP algorithms is diverse, representative and free from biases.
- **Analysis and Detection of bias**: Apply **bias detection** and analysis method on training data to find biases that is based on demographic factors such as race, gender, age.
- **Data Preprocessing:** Data Preprocessing the most important process to train data to mitigate biases like debiasing word embeddings, balance class distributions and augmenting underrepresented samples.
- **Fair representation learning:** Natural Language Processing models are trained to learn fair representations that are invariant to protect attributes like race or gender.
- **Auditing and Evaluation of Models:** Natural Language models are evaluated for fairness and bias with the help of metrics and audits. NLP models are evaluated on diverse datasets and perform post-hoc analyses to find and mitigate innate biases in NLP algorithms.

7. Words with Multiple Meanings

Words with multiple meaning plays a lexical challenge in ***Nature Language Processing*** because of the ambiguity of the word. These words with multiple meaning are known as polysemous or homonymous have different meaning based on the context in which they are used. Here are some key points for representing the lexical challenge plays by words with multiple meanings in NLP:

- **Semantic analysis:** Implement semantic analysis techniques to find the underlying meaning of the word in various contexts. Word embedding or semantic networks are the semantic representation can find the semantic similarity and relatedness between different word sense.
- **Domain specific knowledge:** It is very important to have a specific domain-knowledge in Natural Processing tasks that can be helpful in providing valuable context and constraints for determining the correct context of the word.
- **Multi-word Expression (MWEs):** The meaning of the entire sentence or phrase is analyzed to disambiguate the word with multiple meanings.
- **Knowledge Graphs and Ontologies:** Apply knowledge graphs and ontologies to find the semantic relationships between different words context.

8. Addressing Multilingualism

It is very important to address language diversity and multilingualism in Natural Language Processing to confirm that NLP systems can handle the text data in multiple languages effectively. Here are some key points to address language diversity and multilingualism:

- **Multilingual Corpora:** Multilingual corpus consists of text data in various languages and serve as valuable resources for training NLP models and systems.
- **Cross-Lingual Transfer Learning:** This is a type of techniques that is used to transfer knowledge learned from one language to another.
- **Language Identification:** Design language identification models to automatically detect the language of a given text.
- **Machine Translation:** Machine Translation provides the facility to communicate and inform access across language barriers and can be used as preprocessing step for multilingual NLP tasks.

9. Reducing Uncertainty and False Positives in NLP

It is very crucial task to reduce uncertainty and false positives in Natural Language Process (NLP) to improve the accuracy and reliability of the NLP models. Here are some key points to approach the solution:

- **Probabilistic Models**: Use probabilistic models to figure out the uncertainty in predictions. Probabilistic models such as Bayesian networks gives probabilistic estimates of outputs that allow uncertainty quantification and better decision making.
- **Confidence Scores:** The confidence scores or probability estimates is calculated for NLP predictions to assess the certainty of the output of the model. Confidence scores helps us to identify cases where the model is uncertain or likely to produce false positives.
- **Threshold Tuning:** For the classification tasks the decision thresholds is adjusted to make the balance between sensitivity (recall) and specificity. False Positives in NLP can be reduced by setting the appropriate thresholds.
- **Ensemble Methods:** Apply ensemble learning techniques to join multiple model to reduce uncertainty.

10. Facilitating Continuous Conversations with NLP

- Facilitating continuous conversations with NLP includes the development of system that understands and responds to human language in real-time that enables seamless interaction between users and machines.
- Implementing real time natural language processing pipelines gives to capability to analyze and interpret user input as it is received involving algorithms are optimized and systems for low latency processing to confirm quick responses to user queries and inputs.
- Building an NLP models that can maintain the context throughout a conversation. The understanding of context enables systems to interpret user intent, conversation history tracking, and generating relevant responses based on the ongoing dialogue.
- Apply intent recognition algorithm to find the underlying goals and intentions expressed by users in their messages.

How to overcome NLP Challenges

It requires a combination of innovative technologies, experts of domain, and methodological approached to over the challenges in NLP. Here are some key points to overcome the challenge of NLP tasks:

- **Quantity and Quality of data:** High quality of data and diverse data is used to train the NLP algorithms effectively. Data augmentation, data synthesis, crowdsourcing are the techniques to address data scarcity issues.
- **Ambiguity:** The NLP algorithm should be trained to disambiguate the words and phrases.
- **Out-of-vocabulary Words:** The techniques are implemented to handle out-of-vocabulary words such as tokenization, character-level modeling, and vocabulary expansion.
- **Lack of Annotated Data:** Techniques such transfer learning and pre-training can be used to transfer knowledge from large dataset to specific tasks with limited labeled data.

Spelling error

- One way to deal with spelling errors in NLP is by using techniques such as spell checking, phonetic matching, and incorporating language models that handle out-of-vocabulary words effectively.
- Dealing with spelling errors in Natural Language Processing (NLP) tasks is crucial for improving the accuracy of text-processing applications.
- Here are several techniques commonly used to handle spelling errors:

1. **Spell Checking**:

   - **Dictionary-Based Approaches**: Utilize a dictionary or lexicon to check if each word in the text is spelled correctly. If a word is not found in the dictionary, it is considered a potential spelling error.
   - **Edit Distance Algorithms**: Algorithms such as Levenshtein distance or Damerau-Levenshtein distance measure the minimum number of edits (insertions, deletions, substitutions, or transpositions) required to transform one word into another. Words with small edit distances to known words in the dictionary can be suggested as corrections.

2. Phonetic Matching:

- **Soundex and Metaphone**: Phonetic algorithms map words to phonetic representations based on their pronunciation. Words with similar phonetic representations are likely to be spelled similarly, even if spelled differently. This technique helps in identifying spelling errors where words sound alike but are spelled differently.

3. Language Models:

- **Statistical Language Models**: Use statistical models trained on large text corpora to estimate the probability of a word sequence. Language models can help in identifying likely corrections for misspelled words based on the context of surrounding words.
- **Neural Language Models**: Modern neural language models like Transformer-based models (e.g., BERT, GPT) are effective at predicting and correcting spelling errors by considering the context of the entire sentence. Fine-tuning these models on spelling correction tasks can yield highly accurate results.

4. Rule-Based Approaches:

- **Pattern Matching**: Apply regular expressions or pattern-matching rules to detect common types of spelling errors, such as repeated characters, missing characters, or transposed letters.
- **Language-Specific Rules**: Develop language-specific spelling correction rules based on common misspellings, phonetic patterns, or morphological rules.

5. Ensemble Methods:

- **Combining Multiple Approaches**: Combine the outputs of different spelling correction methods, such as spell checking, phonetic matching, and language models, using ensemble techniques to improve accuracy and robustness.

6. User Feedback:

- **Interactive Correction**: Allow users to provide feedback on suggested corrections and incorporate this feedback to improve the spelling correction system over time. This can be achieved through interactive interfaces or feedback mechanisms in applications.

7. Domain-Specific Customization:

- **Custom Dictionaries**: Create domain-specific dictionaries or lexicons containing relevant terms and vocabulary to improve the accuracy of spelling correction in specific domains or industries.

Spelling correction

- Spelling correction is often considered from two perspectives. Non-word spelling correction is the detection and correction of spelling errors that result in non-words (like graffe for giraffe).
- By contrast, real word spelling correction is the task of detecting and correcting spelling errors even if they accidentally result in an actual word of English (real-word errors).
- This can happen from typographical errors (insertion, deletion, transposition) that accidentally produce a real word (e.g., there for three), or cognitive errors where the writer substituted the wrong spelling of a homophone or near-homophone (e.g., dessert for desert, or piece for peace).
- Non-word errors are detected by looking for any word not found in a dictionary. For example, the misspelling graffe above would not occur in a dictionary. The larger the dictionary the better; modern systems often use enormous dictionaries derived from the web. To correct non-word spelling errors we first generate candidates: real words that have a similar letter sequence to the error.
- Candidate corrections from the spelling error graffe might include giraffe, graf, gaffe, grail, or craft. We then rank the candidates using a distance metric between the source and the surface error.
- We'd like a metric that shares our intuition that giraffe is a more likely source than grail for graffe because giraffe is closer in spelling to graffe than grail is to graffe. The minimum edit distance algorithm will play a role here.

Edit Distance

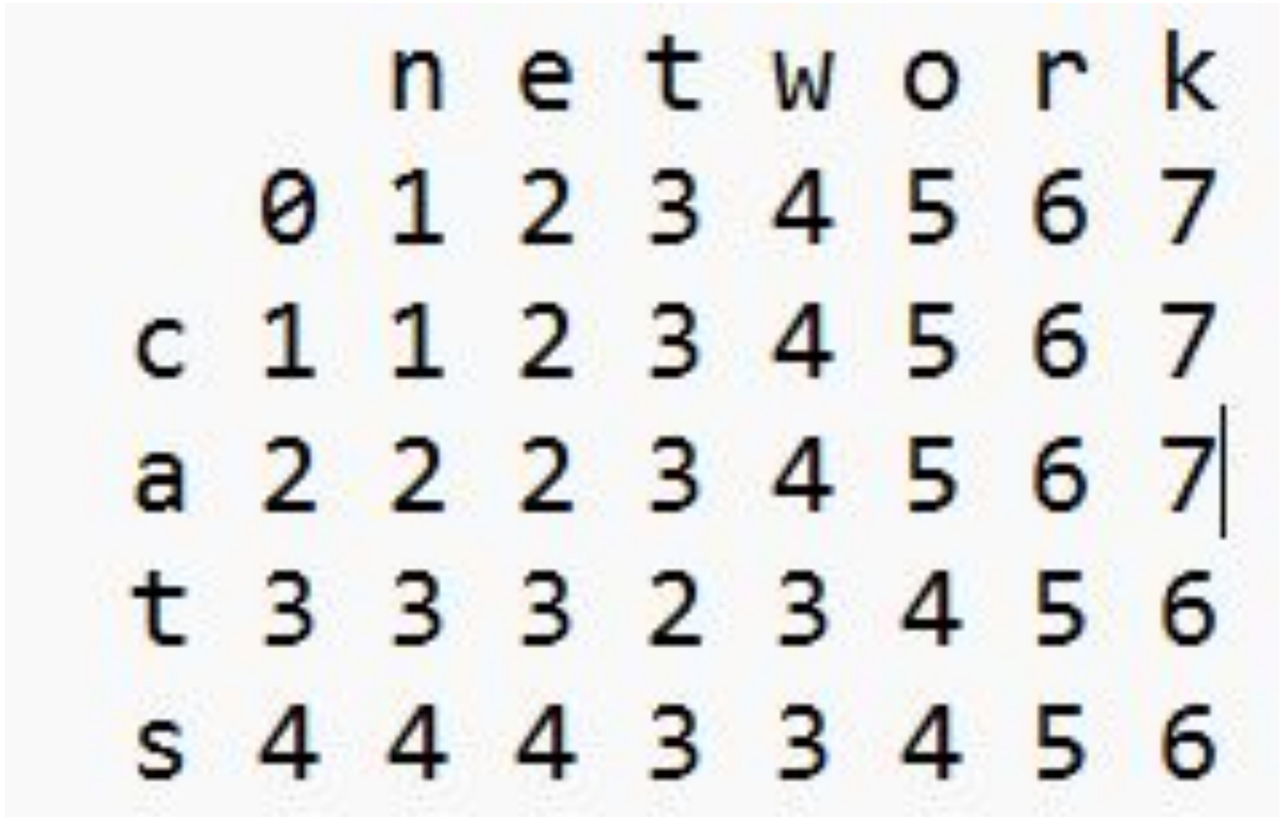
- Given two strings S1 and S2, the minimum number of operations to convert one to the other
- Operations are typically character-level Insert, Delete, Replace, (Transposition)
- E.g., the edit distance from dof to dog is 1
- From cat to act is 2   (Just 1 with transpose.)
- from cat to dog is 3.
- Generally found by dynamic programming.

Find the edit distance from
E=min{E(i-1,j-1)+ if (s1[i]=s2[j]) 0 else 1,   E(i-1,j)+1,   E(i,j-1)+1}

Eg:->network to cats

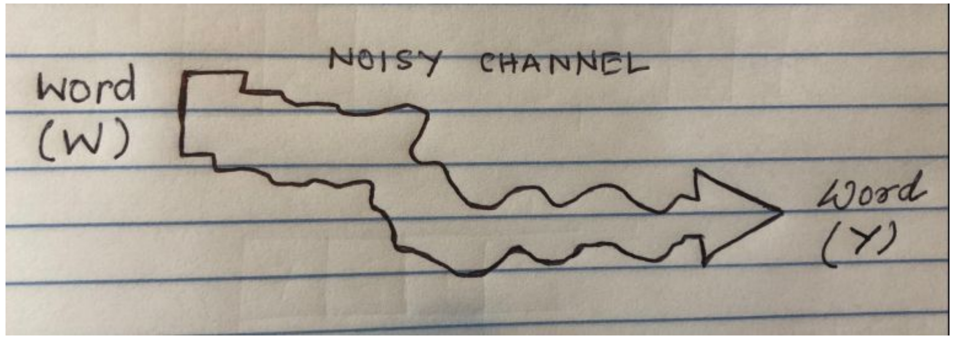
E=min(0+1,1+1,1+1)
=min(1,2,2)
=1

E=min(1+1,1+1,2+1)
=min(2,2,3)
=2



		n	e	t	w	o	r	k
	0	1	2	3	4	5	6	7
c	1	1	2	3	4	5	6	7
a	2	2	2	3	4	5	6	7
t	3	3	3	2	3	4	5	6
s	4	4	4	3	3	4	5	6

Noisy Channel Model



- The Noisy Channel Model is a way to conceptualize many phenomenon or processes in natural language processing/understanding. The applications are myriad: Spelling correction, Question Answering, Machine Translation and many more.
- In this model we assume a piece of text (W) to pass through a noisy channel and we get a garbled text (Y)on the other side of the channel. Or we can also think of it like an encoder that encodes W to Y.
- Now we try to model this noisy channel so that we can find the best estimate of W lets call it W' given Y . Or in the case of the second kind of interpretation we can think of it as modelling a decoder that will decode Y to W' .

- Noisy channel model is a probabilistic model where we use Bayes Rule to determine the probability of the word W' given the word Y .

w' = argmax_w P(w | x)

w' = argmax_w \frac{P(x | w)*P(w)}{P(x)}

w' = argmax_w P(x | w)*P(w)

- Here P(w) is the language model that tells us the probability of the word being w.
- P(x | w) is the noisy channel model that tells us the probability of the original text being w given x.

- Application in Spelling Correction

1. The language model can be chosen as anything a unigram model/ bigram model/ trigram model, with any kind of smoothing.

2. Instead of checking the probability P(x | w) for all words w in the vocabulary we will calculate this probability for only suggested words that are 1 or 2 edit distance from x.

3. For calculating P(x | w) we first build an error confusion matrix that tells us for a pair of letters how likely an edit (deletion/insertion/substitution/transposition) to happen.deletion(x , w) = count(xw converted to x)

   insert(x , w) = count(w converted to xw)

   substitution(x , w) = count(w converted to x)

   transposition(x , w) = count(xw converted to wx)

4. Now to get the probability P(x | w) just divide it by the count of the actual characters:

For deletion: deletion(x , w) / count(xw)

For insertion: insert(x , w) / count(w)

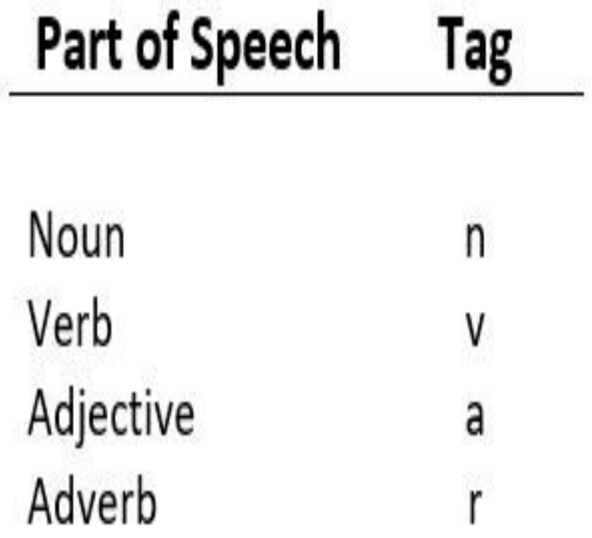
For substitution: substitution(x , w) / count(w)

For transposition: transposition(x , w) / count(xw)

5. Now once we have the P(x | w) for the candidate suggestions (step 2) we select w’ based on w' = argmax_w P(x | w)*P(w) .

Concepts of Parts-of speech

- Parts of Speech tagging is a linguistic activity in Natural Language Processing (NLP) wherein each word in a document is given a particular part of speech (adverb, adjective, verb, etc.) or grammatical category.
- Through the addition of a layer of syntactic and semantic information to the words, this procedure makes it easier to comprehend the sentence's structure and meaning.
- In NLP applications, POS tagging is useful for machine translation, named entity recognition, and information extraction, among other things. It also works well for clearing out ambiguity in terms with numerous meanings and revealing a sentence's grammatical structure.



Part of Speech	Tag
Noun	n
Verb	v
Adjective	a
Adverb	r

- **Default tagging** is a basic step for the part-of-speech tagging. It is performed using the DefaultTagger class. The DefaultTagger class takes ‘tag’ as a single argument. NN is the tag for a singular noun. DefaultTagger is most useful when it gets to work with most common part-of-speech tag. that’s why a noun tag is recommended.

Example of POS Tagging

- Consider the sentence: **“The quick brown fox jumps over the lazy dog.”**
- After performing POS Tagging:

  “The” is tagged as determiner (DT)

  “quick” is tagged as adjective (JJ)

  “brown” is tagged as adjective (JJ)

  “fox” is tagged as noun (NN)

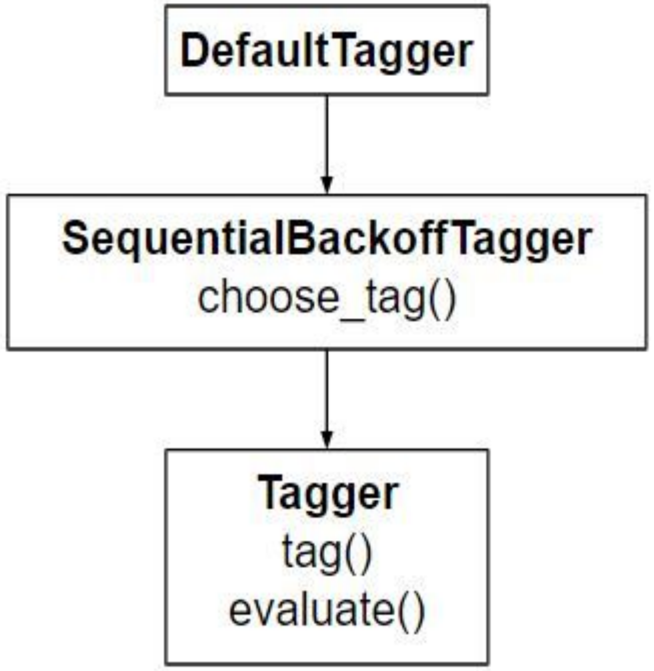
  “jumps” is tagged as verb (VBZ)

  “over” is tagged as preposition (IN)

  “the” is tagged as determiner (DT)

  “lazy” is tagged as adjective (JJ)

  “dog” is tagged as noun (NN)



- By offering insights into the grammatical structure, this tagging aids machines in comprehending not just individual words but also the connections between them inside a phrase. For many NLP applications, like text summarization, sentiment analysis, and machine translation, this kind of data is essential.

Workflow of POS Tagging in NLP

The following are the processes in a typical natural language processing (NLP) example of part-of-speech (POS) tagging:

- **Tokenization:** Divide the input text into discrete tokens, which are usually units of words or subwords. The first stage in NLP tasks is tokenization.
- **Loading Language Models:** To utilize a library such as NLTK or SpaCy, be sure to load the relevant language model. These models offer a foundation for comprehending a language's grammatical structure since they have been trained on a vast amount of linguistic data.
- **Text Processing**: If required, preprocess the text to handle special characters, convert it to lowercase, or eliminate superfluous information. Correct PoS labeling is aided by clear text.
- **Linguistic Analysis**: To determine the text's grammatical structure, use linguistic analysis. This entails understanding each word's purpose inside the sentence, including whether it is an adjective, verb, noun, or other.
- **Part-of-Speech Tagging:** To determine the text's grammatical structure, use linguistic analysis. This entails understanding each word's purpose inside the sentence, including whether it is an adjective, verb, noun, or other.
- **Results Analysis:** Verify the accuracy and consistency of the PoS tagging findings with the source text. Determine and correct any possible problems or mistagging.

- Types of POS Tagging in NLP

Assigning grammatical categories to words in a text is known as Part-of-Speech (PoS) tagging, and it is an essential aspect of Natural Language Processing (NLP). Different PoS tagging approaches exist, each with a unique methodology. Here are a few typical kinds:

1. Rule-Based Tagging

- Rule-based part-of-speech (POS) tagging involves assigning words their respective parts of speech using predetermined rules, contrasting with machine learning-based POS tagging that requires training on annotated text corpora. In a rule-based system, POS tags are assigned based on specific word characteristics and contextual cues.
- For instance, a rule-based POS tagger could designate the “noun” tag to words ending in “-tion” or “-ment,” recognizing common noun-forming suffixes. This approach offers transparency and interpretability, as it doesn’t rely on training data.

Let’s consider an example of how a rule-based part-of-speech (POS) tagger might operate:

Rule: Assign the POS tag “noun” to words ending in “-tion” or “-ment.”

Text: “The presentation highlighted the key achievements of the project’s development.”

Rule based Tags:

“The” – Determiner (DET)

“presentation” – Noun (N)

“highlighted” – Verb (V)

“the” – Determiner (DET)

“key” – Adjective (ADJ)

“achievements” – Noun (N)

“of” – Preposition (PREP)

“the” – Determiner (DET)

“project’s” – Noun (N)

“development” – Noun (N)

In this instance, the predetermined rule is followed by the rule-based POS tagger to label words. “Noun” tags are applied to words like “presentation,” “achievements,” and “development” because of the aforementioned restriction. Despite the simplicity of this example, rule-based taggers may handle a broad variety of linguistic patterns by incorporating different rules, which makes the tagging process transparent and comprehensible.

2. Transformation Based tagging

Transformation-based tagging (TBT) is a part-of-speech (POS) tagging method that uses a set of rules to change the tags that are applied to words inside a text. In contrast, statistical POS tagging uses trained algorithms to predict tags probabilistically, while rule-based POS tagging assigns tags directly based on predefined rules.

To change word tags in TBT, a set of rules is created depending on contextual information. A rule could, for example, change a verb’s tag to a noun if it comes after a determiner like “the.” The text is systematically subjected to these criteria, and after each transformation, the tags are updated.

When compared to rule-based tagging, TBT can provide higher accuracy, especially when dealing with complex grammatical structures. To attain ideal performance, nevertheless, it might require a large rule set and additional computer power.

Consider the transformation rule: Change the tag of a verb to a noun if it follows a determiner like “the.”

Text: “The cat chased the mouse”.

Initial Tags:

“The” – Determiner (DET)

“cat” – Noun (N)

“chased” – Verb (V)

“the” – Determiner (DET)

“mouse” – Noun (N)

Transformation rule applied: Change the tag of “chased” from Verb (V) to Noun (N) because it follows the determiner “the.”

Updated tags:

“The” – Determiner (DET)

“cat” – Noun (N)

“chased” – Noun (N)

“the” – Determiner (DET)

“mouse” – Noun (N)

In this instance, the tag “chased” was changed from a verb to a noun by the TBT system using a transformation rule based on the contextual pattern. The tagging is updated iteratively and the rules are applied sequentially. Although this example is simple, given a well-defined set of transformation rules, TBT systems can handle more complex grammatical patterns.

3. Statistical POS Tagging

Utilizing probabilistic models, statistical part-of-speech (POS) tagging is a computer linguistics technique that places grammatical categories on words inside a text. If rule-based tagging uses massive annotated corpora to train its algorithms, statistical tagging uses machine learning.

In order to capture the statistical linkages present in language, these algorithms learn the probability distribution of word-tag sequences. CRFs (conditional random fields) and Hidden Markov Models (HMMs) are popular models for statistical point-of-sale classification. The algorithm estimates the chance of observing a specific tag given the current word and its context by learning from labeled samples during training.

The most likely tags for text that hasn’t been seen are then predicted using the trained model. Statistical POS tagging works especially well for languages with complicated grammatical structures because it is exceptionally good at handling linguistic ambiguity and catching subtle language trends.

Advantages of POS Tagging

There are several advantages of Parts-Of-Speech (POS) Tagging including:

- **Text Simplification:** Breaking complex sentences down into their constituent parts makes the material easier to understand and easier to simplify.
- **Information Retrieval:** Information retrieval systems are enhanced by point-of-sale (POS) tagging, which allows for more precise indexing and search based on grammatical categories.
- **Named Entity Recognition:** POS tagging helps to identify entities such as names, locations, and organizations inside text and is a precondition for named entity identification.
- **Syntactic Parsing:** It facilitates syntactic parsing, which helps with phrase structure analysis and word link identification.

Disadvantages of POS Tagging

Some common disadvantages in part-of-speech (POS) tagging include:

- **Ambiguity:** The inherent ambiguity of language makes POS tagging difficult since words can signify different things depending on the context, which can result in misunderstandings.
- **Idiomatic Expressions:** Slang, colloquialisms, and idiomatic phrases can be problematic for POS tagging systems since they don’t always follow formal grammar standards.
- **Out-of-Vocabulary Words:** Out-of-vocabulary words (words not included in the training corpus) can be difficult to handle since the model might have trouble assigning the correct POS tags.
- **Domain Dependence:** For best results, POS tagging models trained on a single domain should have a lot of domain-specific training data because they might not generalize well to other domains.

Formal Grammar of English

What is Grammar?

- Grammar is defined as the rules for forming well-structured sentences. Grammar also plays an essential role in describing the syntactic structure of well-formed programs, like denoting the syntactical rules used for conversation in natural languages.
- In the theory of formal languages, grammar is also applicable in Computer Science, mainly in programming languages and data structures.
- Example - In the C programming language, the precise grammar rules state how functions are made with the help of lists and statements.
- Mathematically, a grammar G can be written as a 4-tuple (N, T, S, P) where:

1. N or VN = set of non-terminal symbols or variables.
2. T or ∑ = set of terminal symbols.
3. S = Start symbol where $S \in N$
4. P = Production rules for Terminals as well as Non-terminals.
   It has the form $\alpha \rightarrow \beta$, where α and β are strings on $N \cup \sum$, and at least one symbol of α belongs to VN

Syntax

- Each natural language has an underlying structure usually referred to under Syntax. The fundamental idea of syntax is that words group together to form the constituents like groups of words or phrases which behave as a single unit.
- These constituents can combine to form bigger constituents and, eventually, sentences.
- Syntax describes the regularity and productivity of a language making explicit the structure of sentences, and the goal of syntactic analysis or parsing is to detect if a sentence is correct and provide a syntactic structure of a sentence.

Syntax also refers to the way words are arranged together. Let us see some basic ideas related to syntax:

1. **Constituency:** Groups of words may behave as a single unit or phrase - A constituent, for example, like a Noun phrase.
2. **Grammatical relations:** These are the formalization of ideas from traditional grammar. Examples include - subjects and objects.
3. **Subcategorization and dependency relations:** These are the relations between words and phrases, for example, a Verb followed by an infinitive verb.
4. **Regular languages and part of speech:** Refers to the way words are arranged together but cannot support easily. Examples are Constituency, Grammatical relations, and Subcategorization and dependency relations.
5. **Syntactic categories and their common denotations in NLP:** np - noun phrase, vp - verb phrase, s - sentence, det - determiner (article), n - noun, tv - transitive verb (takes an object), iv - intransitive verb, prep - preposition, pp - prepositional phrase, adj - adjective

Types of Grammar in NLP

Let us move on to discuss the types of grammar in NLP. We will cover three types of grammar: context-free, constituency, and dependency.

Context Free Grammar

- Context-free grammar consists of a set of rules expressing how symbols of the language can be grouped and ordered together and a lexicon of words and symbols.
- One example rule is to express an NP (or noun phrase) that can be composed of either a ProperNoun or a determiner (Det) followed by a Nominal, a Nominal in turn can consist of one or more Nouns:

  NP → DetNominal, NP → ProperNoun; Nominal → Noun | NominalNoun

- Context-free rules can also be hierarchically embedded, so we can combine the previous rules with others, like the following, that express facts about the lexicon:

  Det → a Det → the Noun → flight

- Context-free grammar is a formalism power enough to represent complex relations and can be efficiently implemented. Context-free grammar is integrated into many language applications
- A Context free grammar consists of a set of rules or productions, each expressing the ways the symbols of the language can be grouped, and a lexicon of words

- Context-free grammar (CFG) can also be seen as the list of rules that define the set of all well-formed sentences in a language. Each rule has a left-hand side that identifies a syntactic category and a right-hand side that defines its alternative parts reading from left to right. -
- **Example:** The rule s --> np vp means that "a sentence is defined as a noun phrase followed by a verb phrase."
- Formalism in rules for context-free grammar: A sentence in the language defined by a CFG is a series of words that can be derived by systematically applying the rules, beginning with a rule that has s on its left-hand side.
    - Use of parse tree in context-free grammar: A convenient way to describe a parse is to show its parse tree, simply a graphical display of the parse.
    - A parse of the sentence is a series of rule applications in which a syntactic category is replaced by the right-hand side of a rule that has that category on its left-hand side, and the final rule application yields the sentence itself.

Example: A parse of the sentence "the giraffe dreams" is:

s => np vp

np => det n

np => the n

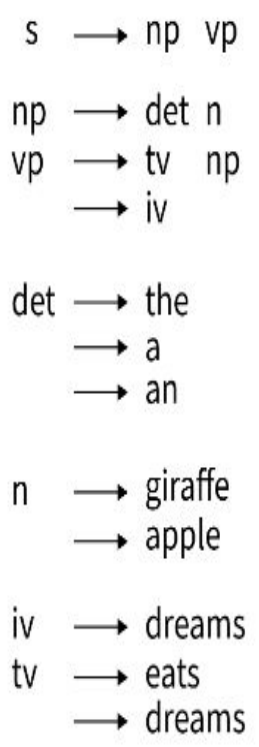
np => the giraffe

vp => iv

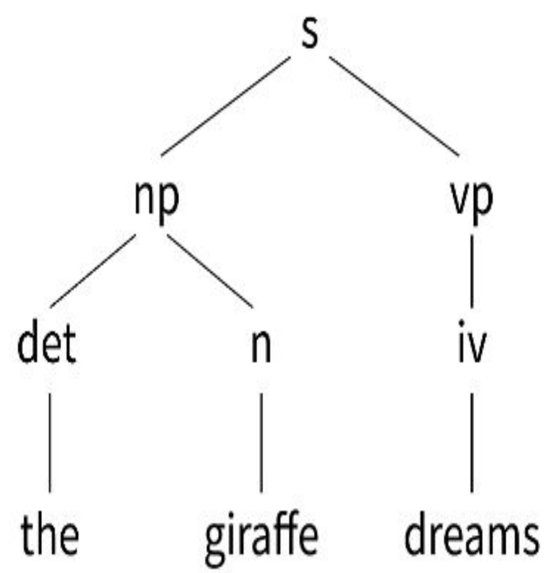
iv=> dreams

iv => the giraffe dreams

- If we look at the example parse tree for the sample sentence in the illustration the giraffe dreams, the graphical illustration shows the parse tree for the sentence
- We can see that the root of every subtree has a grammatical category that appears on the left-hand side of a rule, and the children of that root are identical to the elements on the right-hand side of that rule.



s	$\longrightarrow$	np vp
np	$\longrightarrow$	det n
vp	$\longrightarrow$	tv np
	$\longrightarrow$	iv
det	$\longrightarrow$	the
	$\longrightarrow$	a
	$\longrightarrow$	an
n	$\longrightarrow$	giraffe
	$\longrightarrow$	apple
iv	$\longrightarrow$	dreams
tv	$\longrightarrow$	eats
	$\longrightarrow$	dreams



Classification of Symbols in CFG

The symbols used in Context-free grammar are divided into two classes.

- The symbols that correspond to words in the language, for example, the nightclub, are called terminal symbols, and the lexicon is the set of rules that introduce these terminal symbols.
- The symbols that express abstractions over these terminals are called non-terminals.
- In each context-free rule, the item to the right of the arrow (→) is an ordered list of one or more terminals and non-terminals, and to the left of the arrow is a single non-terminal symbol expressing some cluster or generalization. - The non-terminal associated with each word in the lexicon is its lexical category or part of speech.

Context Free Grammar consists of a finite set of grammar rules that have four components: a Set of Non-Terminals, a Set of Terminals, a Set of Productions, and a Start Symbol.

CFG can also be seen as a notation used for describing the languages, a superset of Regular grammar.

CFG consists of a finite set of grammar rules having the following four components

- **Set of Non-terminals:** It is represented by V. The non-terminals are syntactic variables that denote the sets of strings, which help define the language generated with the help of grammar.
- **Set of Terminals:** It is also known as tokens and is represented by Σ. Strings are formed with the help of the basic symbols of terminals.
- **Set of Productions:** It is represented by P. The set explains how the terminals and non-terminals can be combined.

Every production consists of the following components:

- Non-terminals are also called variables or placeholders as they stand for other symbols, either terminals or non-terminals. They are symbols representing the structure of the language being described. Non-terminals are a set of production rules specifying how to replace a non-terminal symbol with a string of symbols, which can include terminals, words or characters, and other non-terminals.
- Start Symbol: The formal language defined by a CFG is the set of strings derivable from the designated start symbol. Each grammar must have one designated start symbol, which is often called S.
- Since context-free grammar is often used to define sentences, S is usually interpreted as the sentence node, and the set of strings that are derivable from S is the set of sentences in some simplified version of English.

Issues with using context-free grammar in NLP:

- **Limited expressiveness:** Context-free grammar is a limited formalism that cannot capture certain linguistic phenomena such as idiomatic expressions, coordination and ellipsis, and even long-distance dependencies.
- **Handling idiomatic expressions:** CFG may also have a hard time handling idiomatic expressions or idioms, phrases whose meaning cannot be inferred from the meanings of the individual words that make up the phrase.
- **Handling coordination**: CFG needs help to handle coordination, which is linking phrases or clauses with a conjunction.
- **Handling ellipsis:** Context-free grammar may need help to handle ellipsis, which is the omission of one or more words from a sentence that is recoverable from the context.

The limitations of context-free grammar can be mitigated by using other formalisms such as dependency grammar which is powerful but more complex to implement, or using a hybrid approach where both constituency and dependency are used together. We can also additionally use machine learning techniques in certain cases.

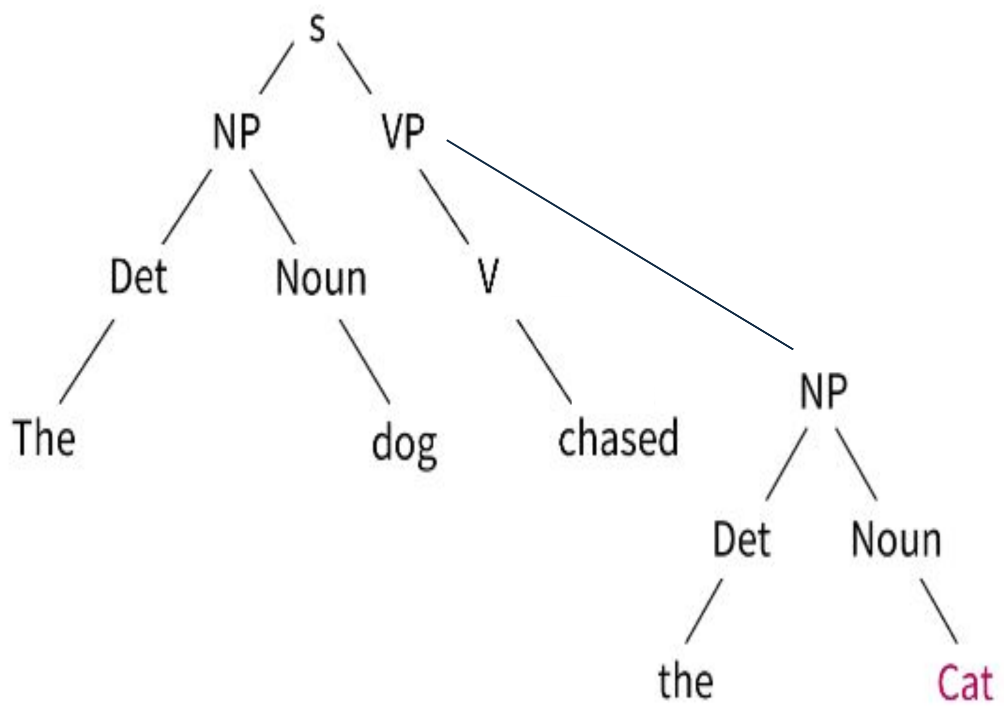
Constituency Grammar

- Constituency Grammar is also known as Phrase structure grammar. Furthermore, it is called constituency Grammar as it is based on the constituency relation. It is the opposite of dependency grammar.
- The constituents can be any word, group of words or phrases in Constituency Grammar. The goal of constituency grammar is to organize any sentence into its constituents using their properties.
- Characteristic properties of constituency grammar and constituency relation:
  - All the related frameworks view the sentence structure in terms of constituency relation.
  - To derive the constituency relation, we take the help of subject-predicate division of Latin as well as Greek grammar.
  - In constituency grammar, we study the clause structure in terms of noun phrase NP and verb phrase VP.
- The properties are derived generally with the help of other NLP concepts like part of speech tagging, a noun or Verb phrase identification, etc. For example, Constituency grammar can organize any sentence into its three constituents - a subject, a context, and an object.

- Look at a sample parse tree: Example sentence -

  "The dog chased the cat."
- In this parse tree, the sentence is represented by the root node S (for sentence). The sentence is divided into two main constituents:

  NP (noun phrase) and VP (verb phrase).
- The NP is further broken down into Det (determiner) and Noun, and the VP is further broken down into V (verb) and NP.
- Each of these constituents can be further broken down into smaller constituents.



Constituency grammar is better equipped to handle context-free and dependency grammar limitations. Let us look at them:

- Constituency grammar is not language-specific, making it easy to use the same model for multiple languages or switch between languages, hence handling the multilingual issue plaguing the other two types of grammar.
- Since constituency grammar uses a parse tree to represent the hierarchical relationship between the constituents of a sentence, it can be easily understood by humans and is more intuitive than other representation grammars.
- Constituency grammar also is simple and easier to implement than other formalisms, such as dependency grammar, making it more accessible for researchers and practitioners.
- Constituency grammar is robust to errors and can handle noisy or incomplete data.
- Constituency grammar is also better equipped to handle coordination which is the linking of phrases or clauses with a conjunction.

Dependency Grammar

Dependency Grammar is the opposite of constituency grammar and is based on the dependency relation. It is opposite to the constituency grammar because it lacks phrasal nodes.

Some fundamental points about Dependency grammar and dependency relation.

- Dependency Grammar states that words of a sentence are dependent upon other words of the sentence. These Words are connected by directed links in dependency grammar. The verb is considered the center of the clause structure.
- Dependency Grammar organizes the words of a sentence according to their dependencies. Every other syntactic unit is connected to the verb in terms of a directed link. These syntactic units are called dependencies.
    - One of the words in a sentence behaves as a root, and all the other words except that word itself are linked directly or indirectly with the root using their dependencies.
    - These dependencies represent relationships among the words in a sentence, and dependency grammar is used to infer the structure and semantic dependencies between the words.

Dependency grammar suffers from some limitations:

- **Ambiguity:** Dependency grammar has issues with ambiguity when it comes to interpreting the grammatical relationships between words, which are particularly challenging when dealing with languages that have rich inflections or complex word order variations.
- **Data annotation:** Dependency parsing also requires labeled data to train the model, which is time-consuming and difficult to obtain.
- **Handling long-distance dependencies:** Dependency parsing also has issues with handling long-term dependencies in some cases where the relationships between words in a sentence may be very far apart, making it difficult to accurately capture the grammatical structure of the sentence.
- **Handling ellipsis and coordination:** Dependency grammar also has a hard time handling phenomena that are not captured by the direct relationships between words, such as ellipsis and coordination, which are typically captured by constituency grammar.

The limitations of dependency grammar can be mitigated by using constituency grammar which, although less powerful, but more intuitive and easier to implement. We can also use a hybrid approach where both constituency and dependency are used together, and it can be beneficial.

Comparing Constituency grammar with Dependency grammar:

- Constituency grammar focuses on grouping words into phrases and clauses, while dependency grammar focuses on the relationships between individual words. Each word in a sentence is represented by a node in a dependency graph, and the edges between nodes represent the grammatical relationships between the words.
  - Dependency grammar is typically more powerful for some languages and NLP tasks as it captures the relationships between the words more accurately, but also more complex to implement and less intuitive.
  - Constituency grammar is more intuitive and easier to implement but can be less expressive.

Language Modelling:

- Language modeling is a crucial element in modern NLP applications and makes the machines understand qualitative information.
- Each language model type, in one way or another, turns the qualitative information generated by humans into quantitative information, which in turn allows people to communicate with machines as they do with each other to a limited extent.
- Language modeling (LM) is the use of various statistical and probabilistic techniques to determine the probability of a given sequence of words occurring in a sentence.
- Language models assign the probabilities to a sentence or a sequence of words or the probability of an upcoming word given a previous set of words.

- Language models are useful for a vast number of NLP applications such as next word prediction, machine translation, spelling correction, authorship Identification, and natural language generation.
- The central idea in language models is to use probability distributions over word sequences that describe how often the sequence occurs as a sentence in some domain of interest.
- Language models estimate the likelihood of texts belonging to a language. The sequences are divided into multiple elements, and the language model models the probability of an element given the previous elements.
  - The elements can be bytes, characters, subwords or tokens. Then, the sequence likelihood is the product of the elements' probabilities.
  - Language models are primarily of two kinds: N-Gram language models and Grammar-based language models such as probabilistic context-free grammar.

- Language models can also be classified into Statistical Language Models and Neural language models.
- Utility of generating language text using language modeling: Independently of any application, we could use a language model as a random sentence generator where we sample sentences according to their language model probability.
    - There are very few real-world use cases where you want to actually generate language randomly.
    - But the understanding of how to do this and what happens when you do so will allow us to do more interesting things later.

Statistical Language Modeling

A statistical language model is simply a probability distribution over all possible sentences. Statistical language models learn the probability of word occurrence based on examples of text.

- While simpler models may look at a context of a short sequence of words, larger statistical language models may function at the level of sentences or paragraphs.
  - Typically, most commonly used statistical language models operate at the level of words.
  - Also, almost all language models decompose the probability of a sentence into a product of conditional probabilities.
- Statistical language models use traditional statistical techniques like N-grams, Hidden Markov Models (HMM), and certain linguistic rules to learn the probability distribution of words.
  - Out of all these models, the most popular and easily implementable are N-gram language models.

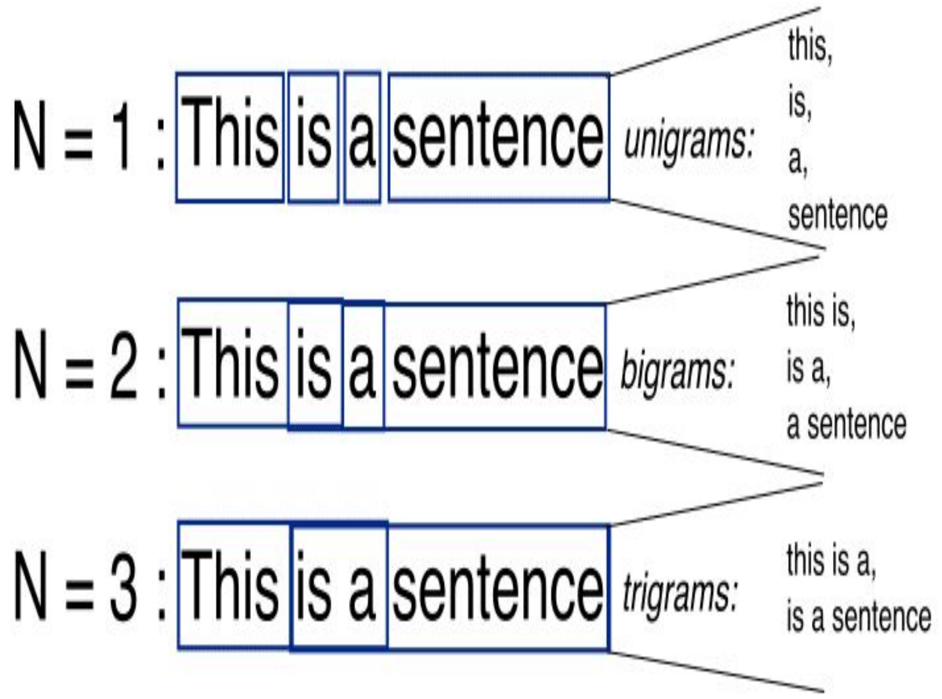
Neural Language Models

- Neural Language Models use different kinds of approaches like neural networks such as feedforward neural networks, recurrent neural nets, attention-based networks, and transformers-based neural nets late to model the language, and they have also surpassed the statistical language models in their effectiveness.
- Neural language models have many advantages over the statistical language models as they can handle much longer histories and also can generalize better over contexts of similar words and are more accurate at word prediction.
- Neural net language models are also much more complex and slower and need more energy to train and are less interpretable than statistical language models.

- Hence for practical purposes where there are not a lot of computing power and training data, and especially for smaller tasks, statistical language models like the n-gram language model are the right tool.
- On the other side, Large language models (LLMs) based on neural networks, in particular, represented state of the art and gave rise to major advancements in NLP AI .
    - They hold the promise of transforming domains through learned knowledge and slowly becoming ubiquitous in day-to-day life related to text and speech use cases
    - LLM sizes have also been increasing 10X every year for the last few years, and these models grow in complexity and size along with their capabilities, like, for example, few shot learners.

N-gram Model

- N-gram models are a particular set of language models based on the statistical frequency of groups of tokens.
- An n-gram is an ordered group of n tokens. The bigrams of the sentence **The cat eats fish.** are (The, cat), (cat, eats), (eats, fish) and (fish, .). The trigrams are (The, cat, eats), (cat, eats, fish) and (eats, fish, .).
- The smallest n-grams with n =1 are called unigrams. Unigrams are simply the tokens appearing in the sentence.
- The conditional probability that a certain token appears after previous tokens are estimated by Maximum Likelihood Estimation on a set of training sequences.



- The intuitive idea behind n-grams and n-gram models is that instead of computing the probability of a word given its entire history, we can approximate the history by just the last few words like humans do while understanding speech and text.
- Illustration for N-gram probabilities

$$\mathbf{P}\left(\mathbf{w}_{\mathrm{n}} \mid \mathbf{w}_{1} \ldots \mathbf{w}_{\mathrm{n}-1}\right) \approx \mathbf{P}\left(\mathbf{w}_{\mathrm{n}}\right) \text{ unigram}$$

$$\mathbf{P}\left(\mathbf{w}_{\mathrm{n}} \mid \mathbf{w}_{1} \ldots \mathbf{w}_{\mathrm{n}-1}\right) \approx \mathbf{P}\left(\mathbf{w}_{\mathrm{n}} \mid \mathbf{w}_{\mathrm{n}-1}\right) \text{ bigram}$$

$$\mathbf{P}\left(\mathbf{w}_{\mathrm{n}} \mid \mathbf{w}_{1} \ldots \mathbf{w}_{\mathrm{n}-1}\right) \approx \mathbf{P}\left(\mathbf{w}_{\mathrm{n}} \mid \mathbf{w}_{\mathrm{n}-1} \mathbf{w}_{\mathrm{n}-2}\right) \text{ trigram}$$

$$\mathbf{P}\left(\mathbf{w}_{\mathrm{n}} \mid \mathbf{w}_{1} \ldots \mathbf{w}_{\mathrm{n}-1}\right) \approx \mathbf{P}\left(\mathbf{w}_{\mathrm{n}} \mid \mathbf{w}_{\mathrm{n}-1} \mathbf{w}_{\mathrm{n}-2} \mathbf{w}_{\mathrm{n}-3}\right) \text{ 4-gram}$$

$$\mathbf{P}\left(\mathbf{w}_{\mathrm{n}} \mid \mathbf{w}_{1} \ldots \mathbf{w}_{\mathrm{n}-1}\right) \approx \mathbf{P}\left(\mathbf{w}_{\mathrm{n}} \mid \mathbf{w}_{\mathrm{n}-1} \mathbf{w}_{\mathrm{n}-2} \mathbf{w}_{\mathrm{n}-3} \mathbf{w}_{\mathrm{n}-4}\right) \text{ 5-gram}$$

- Hence N-Gram models are also a simple class of language models (LM's) that assign probabilities to sequences of words using shorter context.
- We can predict the chance of emerging a word in a given position using these assigned probabilities. For example, the last word of an n-gram given the previous words.

Chain Rule:

$$P(x_1, x_2, \ldots. x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1, x_2) \ldots\ldots P(x_n \mid x_1, \ldots\ldots, x_{n-1})$$

P("about five minutes from") = P(about) × P(five | about) × P(minutes | about five) × P(from | about five minutes)

Probability of words in sentences:

$$P(w_1, w_2, \ldots. w_n) = \prod_i P(w_i \mid w_1, w_2, w_3 \ldots\ldots, w_{i-1})$$

Unigram(1-gram): **No history is used.**

Bi-gram(2-gram): **One word history**

Tri-gram(3-gram): **Two words history**

Four-gram(4-gram): **Three words history**

Five-gram(5-gram):**Four words history**

Generally in practical applications, Bi-gram(previous one word), Tri-gram(previous two word, Four-gram (previous three word) are used.

Unigram(1-gram): No history is used.

“about five minutes from.....”

Assume in corpus dinner word is present with highest probability.

Unigram doesn’t take into account probabilities with previous words like from, minutes.

Unigram will predict dinner.

“about five minutes from dinner”

Bi-gram(2-gram): **One word history**

$$P(w_1, w_2) = \prod_{i=2} P(w_2 \mid w_1) \qquad P(w_i \mid w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

"about five minutes from....."

Assumption: Next word may be college, class

$$P(\text{college} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from college)}}{\text{count(about five minutes from)}}$$

$$P(\text{class} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from class)}}{\text{count(about five minutes from)}}$$

"about five minutes from....."

Count(about five minutes from)= P(about|<S>) × P(five| about) × P(minutes | five) × P(from| minutes)

Count(about five minutes from **college**)= P(about|<S>) × P(five| about) × P(minutes | five) × P(from| minutes) × P(college| from)

Count(about five minutes from **class**)= P(about|<S>) × P(five| about) × P(minutes | five) × P(from| minutes) × P(class| from)

$$P(\text{college} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from college)}}{\text{count(about five minutes from)}}$$

=P(college| from)

$$P(\text{class} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from class)}}{\text{count(about five minutes from)}}$$

=P(class| from)

Tri-gram(2-gram): **Two words history**

$$P(w_1, w_2, w_3) = \prod_{i=3} P(w_3 \mid w_1, w_2) \qquad P(w_i \mid w_{i-1}, w_{i-2}) = \frac{\text{count}(w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-2}, w_{i-1})}$$

Count(about five minutes from)= P(five |<S>, about) × P(minutes| about, five) × P(from| five , minutes)

Count(about five minutes from **college**)= P(five |<S>, about) × P(minutes| about, five) × P(from| five , minutes) × P(college| minutes from)

Count(about five minutes from **class**)= P(five |<S>, about) × P(minutes| about, five) × P(from| five , minutes) × P(class| minutes from)

$$P(\text{college} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from college)}}{\text{count(about five minutes from)}}$$

=P(college| minutes from)

$$P(\text{class} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from class)}}{\text{count(about five minutes from)}}$$

=P(class| minutes from)

Four-gram(4-gram): **Three words history**

$$P(w_1, w_2, w_3, w_4) = \prod_{i=4} P(w_4 \mid w_1, w_2, w_3)$$

$$P(w_i \mid w_{i-1}, w_{i-2}, w_{i-3}) = \frac{\text{count}(w_{i-3}, w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-3}, w_{i-2}, w_{i-1})}$$

Count(about five minutes from)= P(minutes|<S>, about, five) × P(from| about, five, minutes)

Count(about five minutes from **college**)= P(minutes|<S>, about, five) ×
P(from| about, five, minutes) ×
P(college| five , minutes, from)

Count(about five minutes from **class**)= P(minutes|<S>, about, five) ×
P(from| about, five, minutes) ×
P(class| five , minutes, from)

$$P(\text{college} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from college)}}{\text{count(about five minutes from)}}$$

=P(college | five minutes from)

$$P(\text{class} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from class)}}{\text{count(about five minutes from)}}$$

=P(college | five minutes from)

As no. of previous state (history) increases, it is very difficult to match that set of words in corpus.

Probabilities of **larger collection of word is minimum.** To overcome this problem, Bi-gram model is used

Exercise 1: Estimating Bi-gram probabilities

What is the most probable next word predicted by the model for the following word sequence?

Given Corpus

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Henry I am </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

<S> I like Henry ?

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Next word prediction probability **W_{i-1}=Henry**

Next word	Probability Next Word= $\frac{N}{D} = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$
P(</S> | Henry)	**3/5**
P(<I> | Henry)	1/5
P(<am>| Henry)	0
P(<Henry>| Henry)	0
P(<like | Henry)	1/5
P(<college | Henry)	0
P(do | Henry)	0

</S> is more probable

Which of the following sentence is better. i.e. Gets a higher probability with this model. Use Bi-gram

<S> I am Henry </S>	
<S> I like college </S>	
<S> Do Henry like college </S>	
<S> Henry I am </S>	
<S> Do I like Henry </S>	
<S> Do I like college </S>	
<S> I do like Henry </S>	

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

1. <S> I like college </S>

<S> like college </S>=?

=P(I| <S>) × P(like | I) × P(college| like) × P(</S> | college)

=3/7 × 3/6 × 3/5 ×3/3 = 9/70=0.13

2. <S> Do I like Henry </S>

=P(do | <S>) × P(I | do) × P(like | I) × P(Henry | like) × P(</S> | Henry)

=3/7 × 2/4 × 3/6 ×2/5 ×3/5 = 9/350=0.0257

ANS: First statement is more probable

<S> Do I like ?

Use Tri-gram

P<I like>=3

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Henry I am </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

Next word prediction probability

W_{i-2}=I and W_{i-1}=like

Next word	Probability Next Word= $\frac{\text{count}(w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-2}, w_{i-1})}$
P(</S> |I like)	0/3
P(<I> | I like)	0/3
P(<am>| I like)	0/3
P(<Henry>| I like)	1/3
P(<like | I like)	0/3
P(<college | I like)	2/3
P(do | I like)	0/3

College is probable

<S> Do I like college ?

Use Four-gram

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Next word prediction probability

W_{i-3}=I, W_{i-2}=like W_{i-1}=college

Next word	Probability Next Word= $\frac{\text{count}(w_{i-3}, w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-3}, w_{i-2}, w_{i-1})}$
P(</S> |I like college)	2/2
P(<I> | I like college)	0/2
P(<am>| I like college)	0/2
P(<Henry>| I like college)	0/2
P(<like | I like college)	0/2
P(<college | I like college)	0/2
P(do | I like college)	0/2

</S> is more probable

<s> the traffic lights </s>

<s> traffic lights switched to red</s>

<s> lights switched from red to green</s>

<s> red lights switched to green </s>

<s> from green to red</s>

<s> green to red </s>

using bi-gram find probability for

1) <s> the traffic lights ?
2) red ?
3) <s> green ?

Evaluating language models

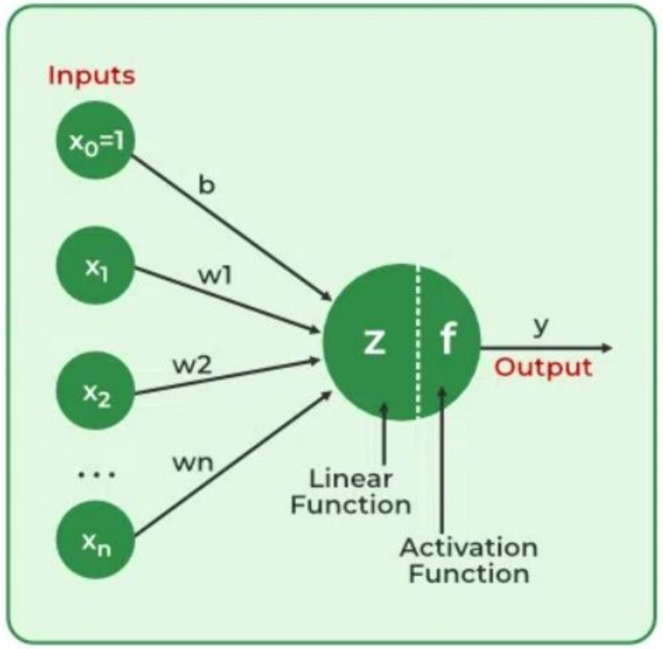
- A machine learning model, in general, is considered good if it predicts a test set of sentences.
    - We reserve some portion of the data for estimating parameters, and we use the remainder for testing the model.
    - A good model assigns high probabilities to the test sentences where typically the probability of each sentence is normalized for length.
- An alternative approach that measures how well the test samples are predicted is using the measure of perplexity.
    - Models that minimize perplexity will also maximize the probability.
    - When we look at the mathematical formulation for perplexity, it is simply the inverse of the probability with a log transform applied on top of it.
- Perplexity: Perplexity is a measure of surprise in random choices.
    - Distributions with high uncertainty have high perplexity. A uniform distribution has high perplexity because it is hard to predict a random draw from it.
    - A peaked distribution has low perplexity because it is easy to predict the outcome of a random draw from it.

Neural Network basics

Neural networks extract identifying features from data, lacking pre-programmed understanding. Network components include neurons, connections, weights, biases, propagation functions, and a learning rule. Neurons receive inputs, governed by thresholds and activation functions. Connections involve weights and biases regulating information transfer. Learning, adjusting weights and biases, occurs in three stages: **input computation, output generation, and iterative refinement** enhancing the network's proficiency in diverse tasks.

These include:

- The neural network is simulated by a new environment.
- Then the free parameters of the neural network are changed as a result of this simulation.
- The neural network then responds in a new way to the environment because of the changes in its free parameters.



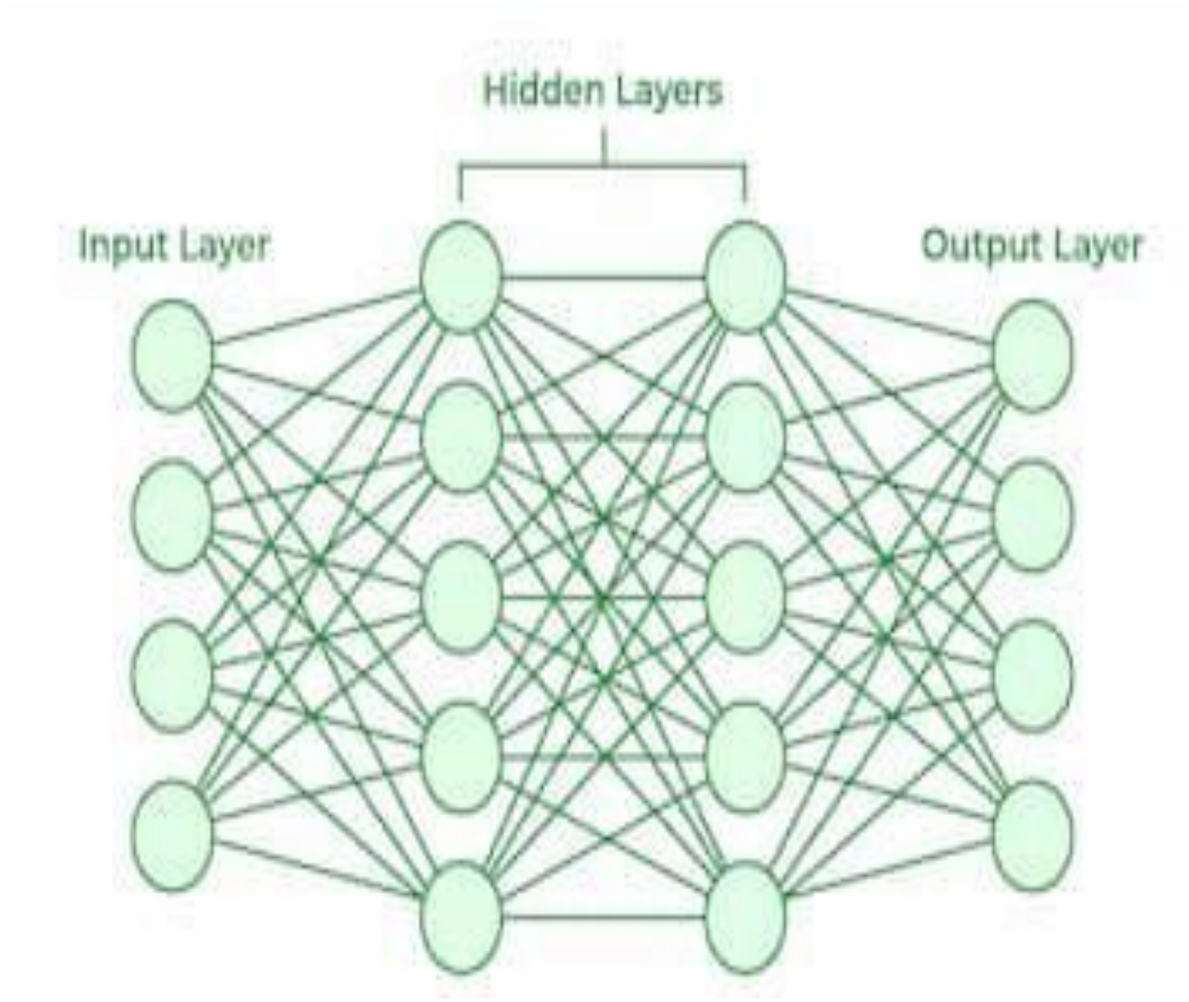
How does Neural Networks work?

Let’s understand with an example of how a neural network works:

- Consider a neural network for email classification. The input layer takes features like email content, sender information, and subject.
- These inputs, multiplied by adjusted weights, pass through hidden layers. The network, through training, learns to recognize patterns indicating whether an email is spam or not.
- The output layer, with a binary activation function, predicts whether the email is spam (1) or not (0).
- As the network iteratively refines its weights through backpropagation, it becomes adept at distinguishing between spam and legitimate emails, showcasing the practicality of neural networks in real-world applications like email filtering.

Working of a Neural Network

- Neural networks are complex systems that mimic some features of the functioning of the human brain.
- It is composed of an input layer, one or more hidden layers, and an output layer made up of layers of artificial neurons that are coupled.
- The two stages of the basic process are called backpropagation and forward propagation.



Forward Propagation

- Input Layer: Each feature in the input layer is represented by a node on the network, which receives input data.
- Weights and Connections: The weight of each neuronal connection indicates how strong the connection is. Throughout training, these weights are changed.
- Hidden Layers: Each hidden layer neuron processes inputs by multiplying them by weights, adding them up, and then passing them through an activation function. By doing this, non-linearity is introduced, enabling the network to recognize intricate patterns.
- Output: The final result is produced by repeating the process until the output layer is reached.

Backpropagation

- Loss Calculation: The network’s output is evaluated against the real goal values, and a loss function is used to compute the difference. For a regression problem, the Mean Squared Error (MSE) is commonly used as the cost function.
- Loss Function:MSE = \frac{1}{n} \Sigma^{n}_{i=1} (y_{i} - \hat y_{i})^2
- Gradient Descent: Gradient descent is then used by the network to reduce the loss. To lower the inaccuracy, weights are changed based on the derivative of the loss with respect to each weight.
- Adjusting weights: The weights are adjusted at each connection by applying this iterative process, or backpropagation, backward across the network.
- Training: During training with different data samples, the entire process of forward propagation, loss calculation, and backpropagation is done iteratively, enabling the network to adapt and learn patterns from the data.
- Actvation Functions: Model non-linearity is introduced by activation functions like the rectified linear unit (ReLU) or sigmoid. Their decision on whether to “fire” a neuron is based on the whole weighted input.

Morphology & Parsing in NLP:

Computational Morphology

Morphemes

- Morphological theories differ on whether and how to associate the properties of word forms with their structural components.
- These components are usually called segments or morphs.
- The morphs that by themselves represent some aspect of the meaning of a word are called morphemes of some function.
- Human languages employ a variety of devices by which morphs and morphemes are combined into word forms.

Morphology

- Morphology is the domain of linguistics that analyses the internal structure of words.
- Morphological analysis – exploring the structure of words
- Words are built up of minimal meaningful elements called morphemes:

  played = play-ed

  cats = cat-s

  unfriendly = un-friend-ly
- Two types of morphemes:

  i Stems: play, cat, friend

  ii Affixes: -ed, -s, un-, -ly
- Two main types of affixes:

  i Prefixes precede the stem: un-, re-

  ii Suffixes follow the stem: -ed, -s, -ly
- Stemming = find the stem by stripping off affixes

  play = play

  replayed = re-play-ed

  computerized = comput-er-ize-d

Problems in morphological processing

- **Inflectional morphology:** inflected forms are constructed from base forms and inflectional affixes.
- Inflection relates different forms of the same word

Lemma	Singular	Plural
cat	cat	cats
dog	dog	dogs
knife	knife	knives
sheep	sheep	sheep
mouse	mouse	mice

- **Derivational morphology:** words are constructed from roots (or stems) and derivational affixes:

inter+national = international

international+ize = internationalize

internationalize+ation = internationalization

Morphological Parsing

Requirement to build a Morphological Parser

1. **a lexicon:** The list of stem and affixes, together with basic information about them
2. **morphotactics:** The model of morpheme ordering that explains which classes of morphemes can follow other classes of morphemes inside a word

   for example: Rule that English plural morphemes follow the noun rather than preceding it
3. **orthographic rules:** these spelling rules are used to model the changes that occur in a word, usually when two morphemes combine

   for example: spelling rule that changes city+s to cities rather than citys

- English has concatenative morphology.
- Words can be made up of main stem (carrying the basic dictionary meaning) plus one or more affixes carrying grammatical information.
  - Surface form: cats walking smoothest
  - Lexical form: cat+N+PL walk+V+PresPart smooth+Adj+sup
- Morphological Parsing is the problem of extracting the lexical form from the surface form
- Consider following example:

INPUT	**Morphological parsed OUTPUT**
cities	city + N + PL
merging	merge + V + Pres-Part
geese	goose + N + PL

Morphological Analysis:

- It includes identification, analysis and description of the structure of a given languages. Morphemes and other linguistics units such as root words, affixes, part of speech, intonation and stresses or implied context
- It helps detect Ambiguity ie. more than one alternative
- For example:

  flies lexical form can be **fly + V + Prog** or **fly + N + PL**

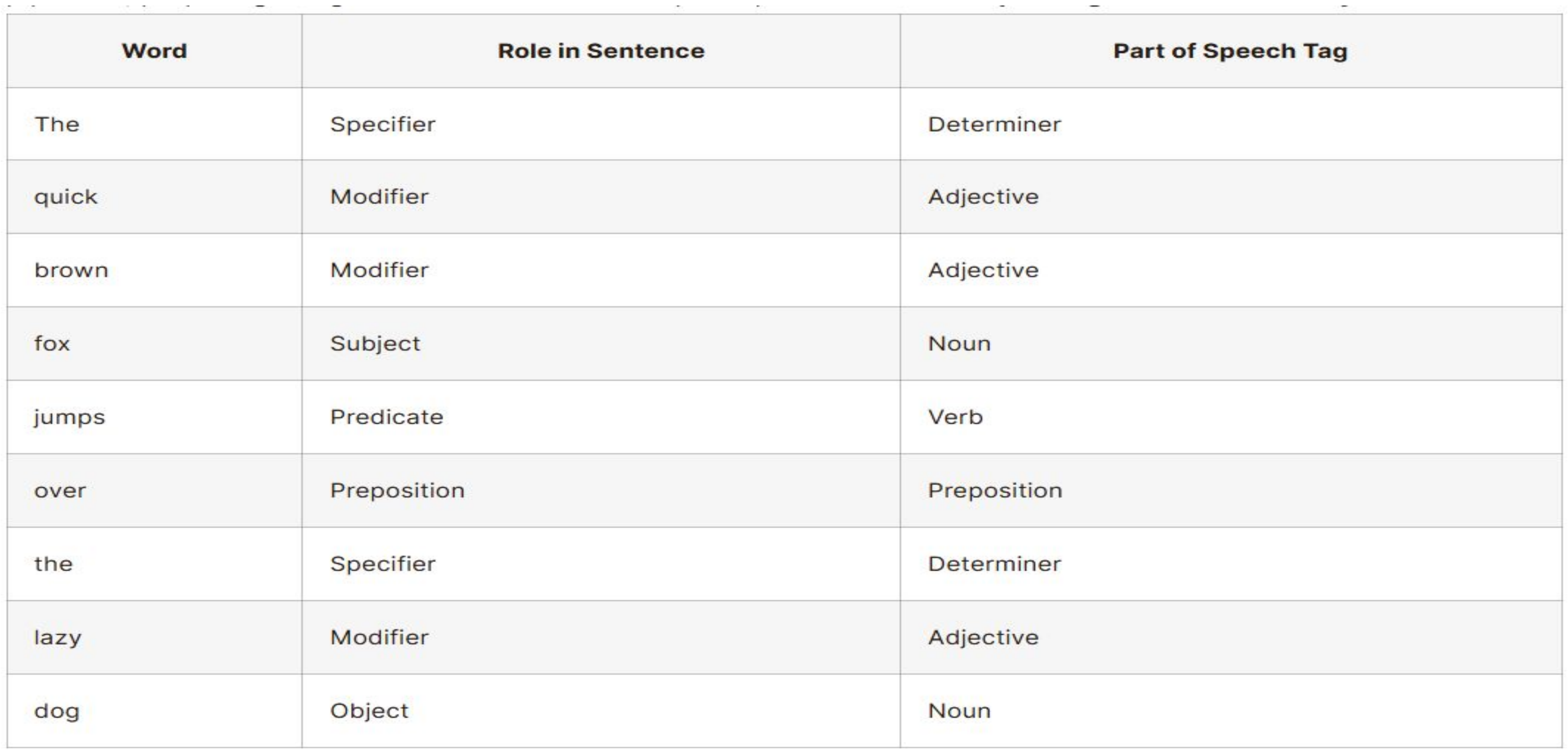
Need of Morphological Analysis:

- Wastage of memory in exhaustive lexicon
- Failure to depict linguistic generalization necessary to understand an unknown word
- Morphologically rich and productive language might be problematic

Parts-of-speech Tagging: Basic Concepts

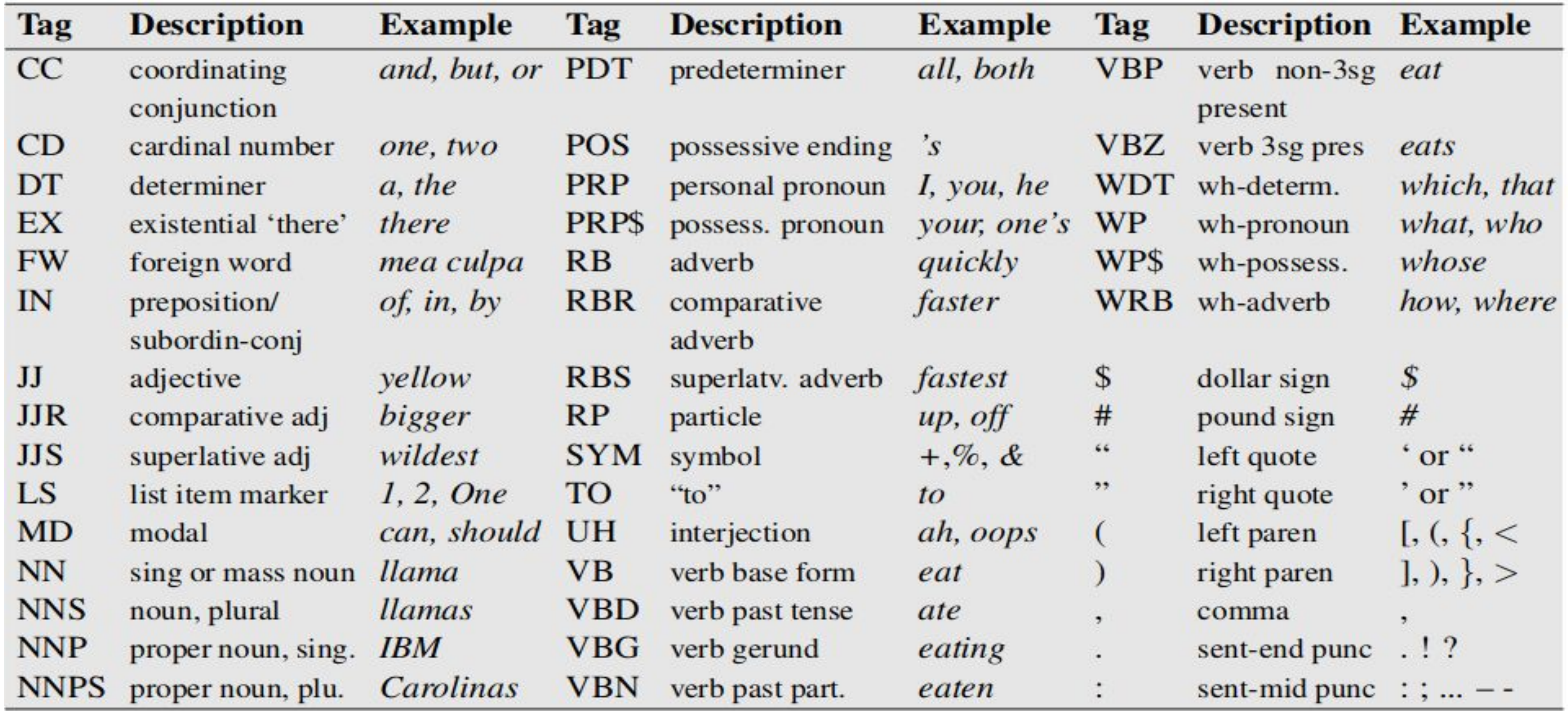
- The world of Part-of-Speech Tagging Basics in Natural Language Processing (NLP), is the critical underpinning of how we enable computers to parse, understand, and respond to human language with precision.
- A system where every word in a given text is meticulously labeled with its corresponding part of speech—a noun, a verb, an adjective, and so on. This process, known as Part-of-Speech (POS) tagging, involves rigorously categorizing words based on their definitions and context within a sentence. Achieving this allows for a deeper syntactic understanding of language – a process essential for machines to process human language accurately.
- The tagging algorithm acts like a master linguist, sifting through the complexities of syntax to lay bare the grammatical framework upon which meaning pivots. Quite often, POS tagging is one of the preliminary steps in NLP pipelines, preparing the groundwork for more complex operations such as parsing or semantic analysis.

For example:



Word	Role in Sentence	Part of Speech Tag
The	Specifier	Determiner
quick	Modifier	Adjective
brown	Modifier	Adjective
fox	Subject	Noun
jumps	Predicate	Verb
over	Preposition	Preposition
the	Specifier	Determiner
lazy	Modifier	Adjective
dog	Object	Noun

POS Tagsets



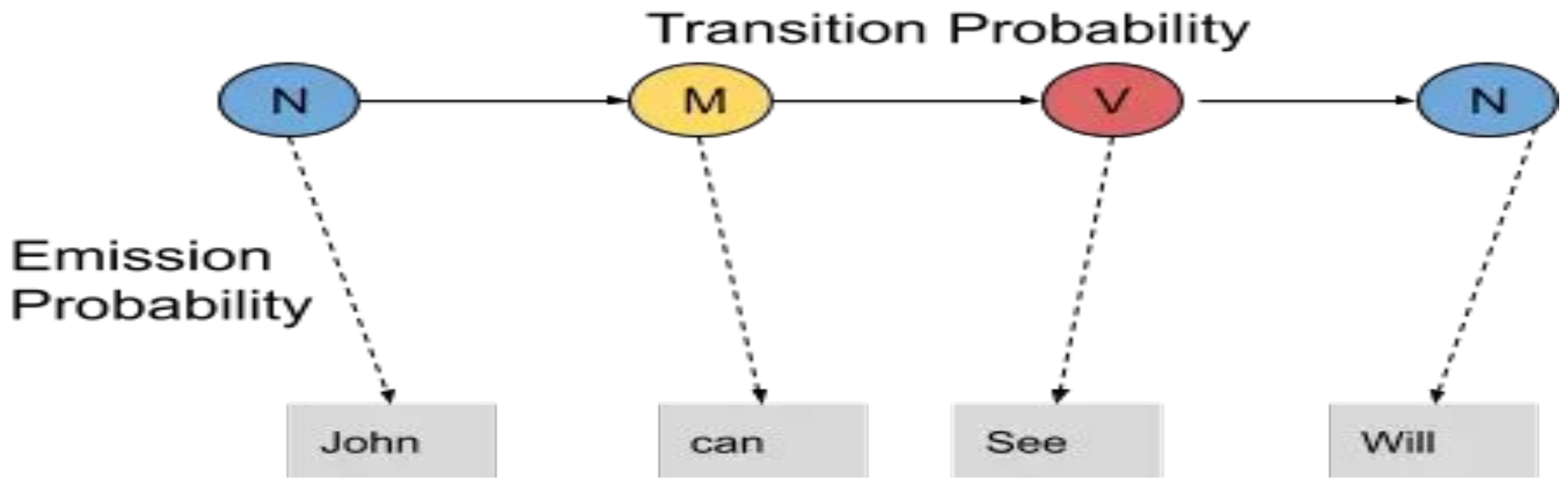
Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coordinating conjunction	*and, but, or*	PDT	predeterminer	*all, both*	VBP	verb non-3sg present	*eat*
CD	cardinal number	*one, two*	POS	possessive ending	*'s*	VBZ	verb 3sg pres	*eats*
DT	determiner	*a, the*	PRP	personal pronoun	*I, you, he*	WDT	wh-determ.	*which, that*
EX	existential 'there'	*there*	PRP$	possess. pronoun	*your, one's*	WP	wh-pronoun	*what, who*
FW	foreign word	*mea culpa*	RB	adverb	*quickly*	WP$	wh-possess.	*whose*
IN	preposition/ subordin-conj	*of, in, by*	RBR	comparative adverb	*faster*	WRB	wh-adverb	*how, where*
JJ	adjective	*yellow*	RBS	superlatv. adverb	*fastest*	$	dollar sign	*$*
JJR	comparative adj	*bigger*	RP	particle	*up, off*	#	pound sign	*#*
JJS	superlative adj	*wildest*	SYM	symbol	*+,%, &*	“	left quote	‘ or “
LS	list item marker	*1, 2, One*	TO	“to”	*to*	”	right quote	’ or ”
MD	modal	*can, should*	UH	interjection	*ah, oops*	(	left paren	[, (, {, <
NN	sing or mass noun	*llama*	VB	verb base form	*eat*	)	right paren	],), }, >
NNS	noun, plural	*llamas*	VBD	verb past tense	*ate*	,	comma	,
NNP	proper noun, sing.	*IBM*	VBG	verb gerund	*eating*	.	sent-end punc	. ! ?
NNPS	proper noun, plu.	*Carolinas*	VBN	verb past part.	*eaten*	:	sent-mid punc	: ; ... – -

Lemmatization

- Lemmatization is the process of grouping together word forms that belong to the same inflectional paradigm and assigning to each paradigm its corresponding uninflected form, called a ‘lemma’.
- This form of annotation is related to the process of part-of-speech tagging, as the latter is a prerequisite for lemmatization.
- Lemmatization is regarded as a highly useful type of corpus annotation because it provides additional search options.
- The lemmatization of VOICE was carried out by an especially designed lemmatizer which was implemented in Python. This lemmatizer accessed the VOICE Lexicon in its final, completed version and retrieved the appropriate lemma from there.
- The lemmatizer did this by applying a number of manually implemented rules, which were related to the information contained in the POS tags (e.g. for regular verb forms: if tag for word X is VV (verb, base form), then lemma equals token), or, sometimes, also to the morphological information contained in the tokens (e.g. for regular adjective forms: if tag for word X is JJR (adjective, comparative), then lemma equals token without the suffix - er).

POS tagging using HMM

- HMM (Hidden Markov Model) is a Stochastic technique for POS tagging.
- HMM are known for their applications to reinforcement learning and temporal pattern recognition such as speech, handwriting, gesture recognition, musical score following, partial discharges, and bioinformatics.
- Let us consider an example proposed by Dr.Luis Serrano and find out how HMM selects an appropriate tag sequence for a sentence.

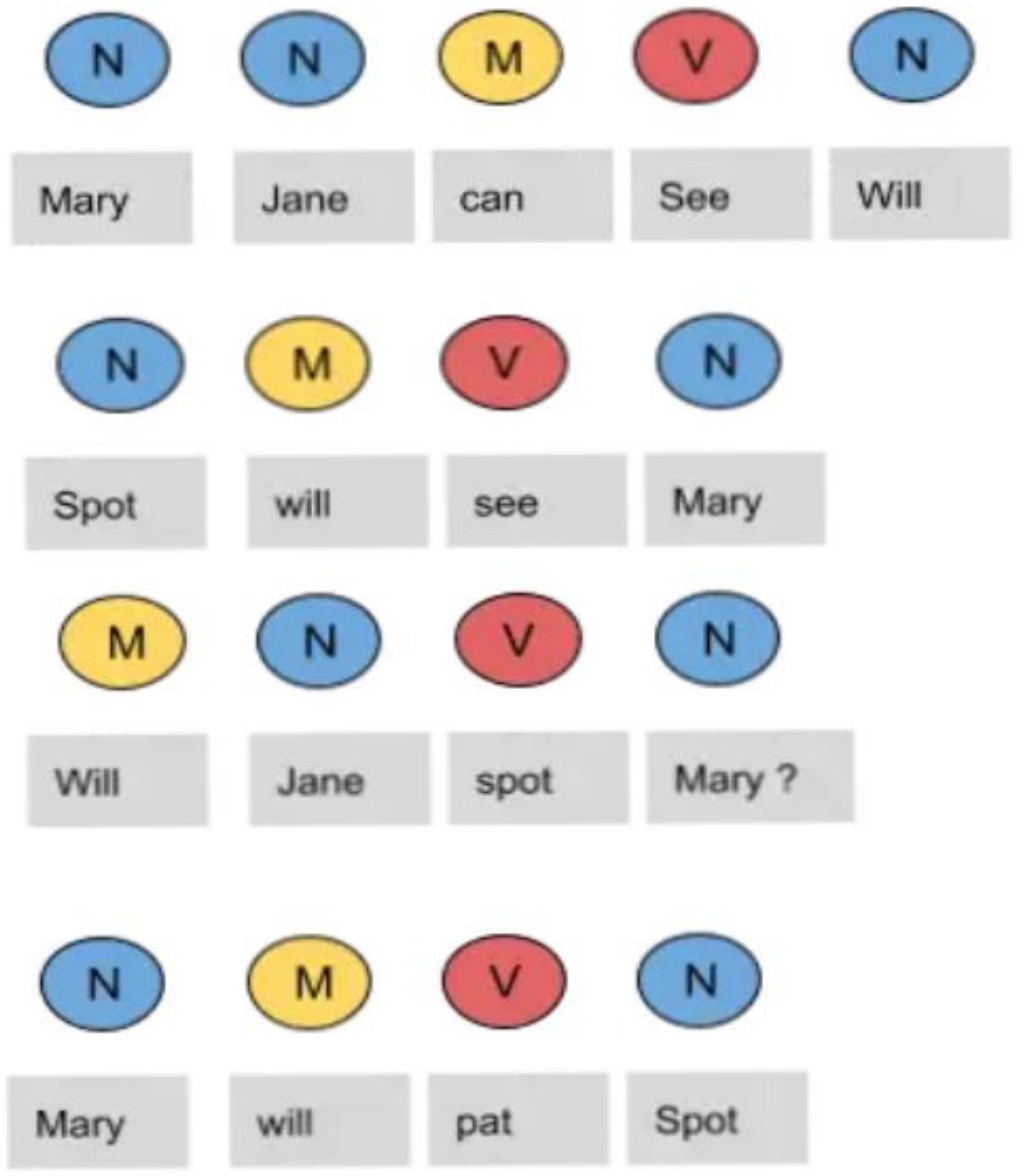


- In the above example, we consider only 3 POS tags that are noun, model and verb.
- Let the sentence be “ Ted will spot Will ” be tagged as noun, model, verb and a noun and to calculate the probability associated with this particular sequence of tags we require their Transition probability and Emission probability.
- The transition probability is the likelihood of a particular sequence for example, how likely is that a noun is followed by a model and a model by a verb and a verb by a noun. This probability is known as Transition probability. It should be high for a particular sequence to be correct.
- Now, what is the probability that the word Ted is a noun, will is a model, spot is a verb and Will is a noun. These sets of probabilities are Emission probabilities and should be high for our tagging to be likely.

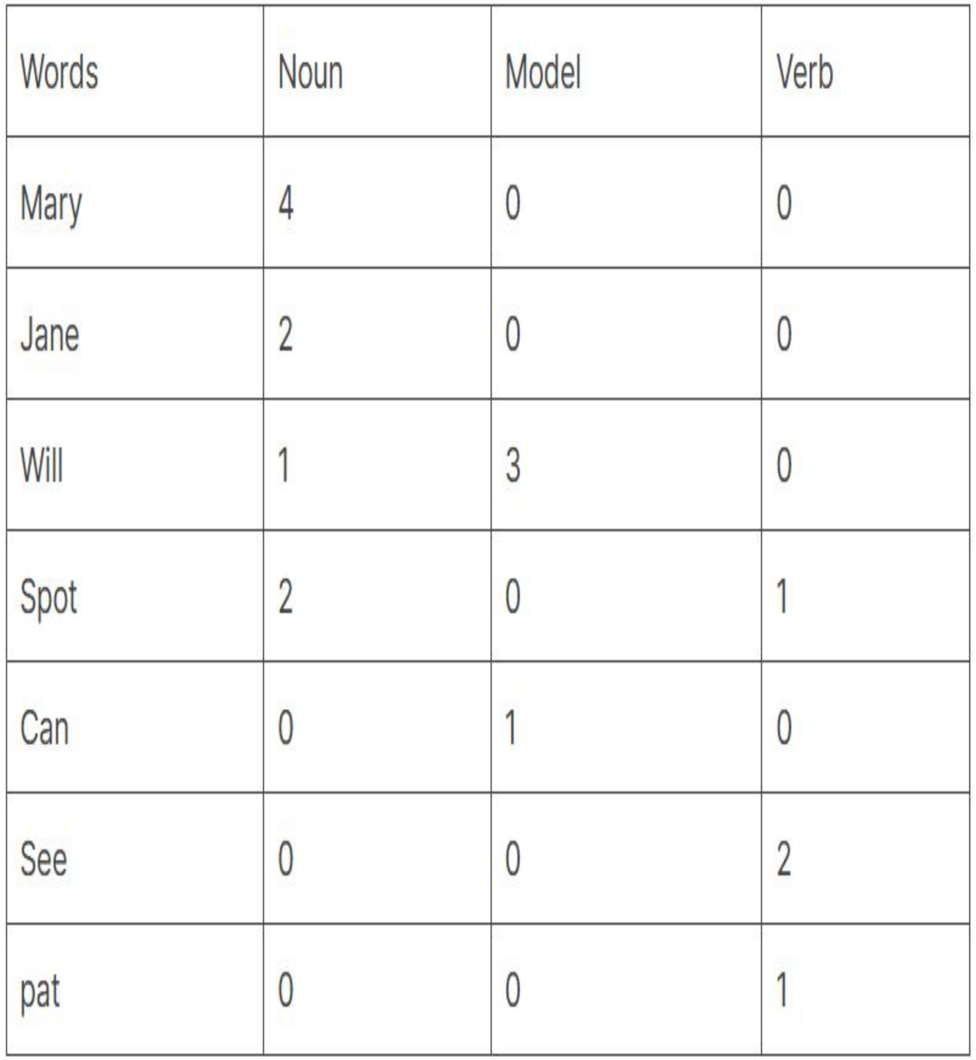
- Let us calculate the above two probabilities for the set of sentences below

1. Mary Jane can see Will
2. Spot will see Mary
3. Will Jane spot Mary?
4. Mary will pat Spot

- Note that Mary Jane, Spot, and Will are all names.

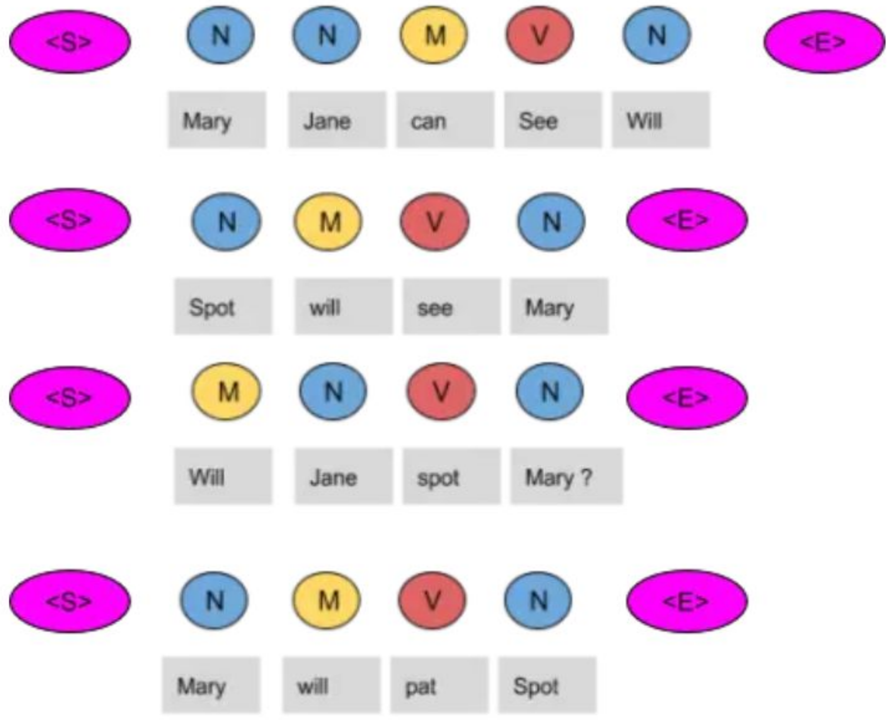


- In the above sentences, the word Mary appears four times as a noun.
- To calculate the emission probabilities, let us create a counting table in a similar manner.
- From the table, we infer that
  - The probability that Mary is Noun = 4/9
  - The probability that Mary is Model = 0
  - The probability that Will is Noun = 1/9
  - The probability that Will is Model = 3/4
- In a similar manner, you can figure out the rest of the probabilities. These are the emission probabilities.



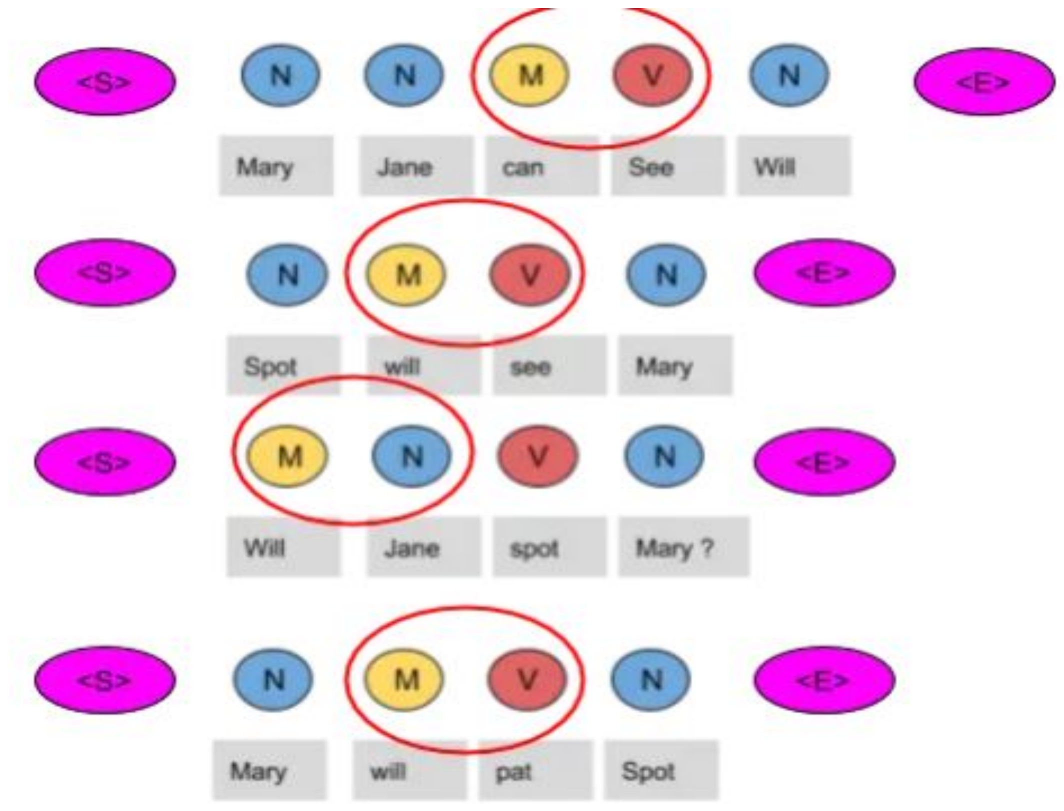
Words	Noun	Model	Verb
Mary	4	0	0
Jane	2	0	0
Will	1	3	0
Spot	2	0	1
Can	0	1	0
See	0	0	2
pat	0	0	1

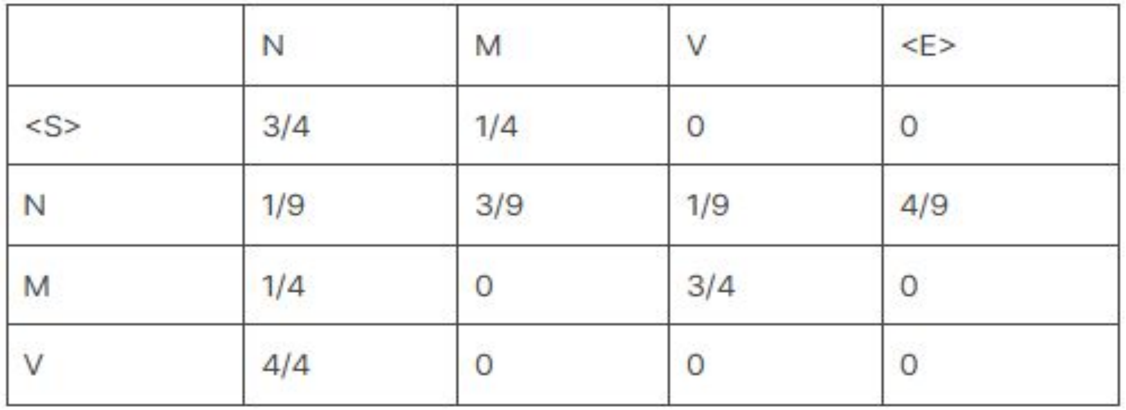
- Next, we have to calculate the transition probabilities, so define two more tags <S> and <E>. <S> is placed at the beginning of each sentence and <E> at the end as shown in the figure.
- Let us again create a table and fill it with the co-occurrence counts of the tags.
- In the figure, we can see that the <S> tag is followed by the N tag three times, thus the first entry is 3.The model tag follows the <S> just once, thus the second entry is 1. In a similar manner, the rest of the table is filled.



	N	M	V	<E>
<S>	3	1	0	0
N	1	3	1	4
M	1	0	3	0
V	4	0	0	0

- Next, we divide each term in a row of the table by the total number of co-occurrences of the tag in consideration, for example, The Model tag is followed by any other tag four times as shown below, thus we divide each element in the third row by four.
- These are the respective transition probabilities for the above four sentences.
- Now how does the HMM determine the appropriate sequence of tags for a particular sentence from the above tables? Let us find it out.





	N	M	V	<E>
<S>	3/4	1/4	0	0
N	1/9	3/9	1/9	4/9
M	1/4	0	3/4	0
V	4/4	0	0	0

- Take a new sentence and tag them with wrong tags. Let the sentence, ‘ Will can spot Mary’ be tagged as-

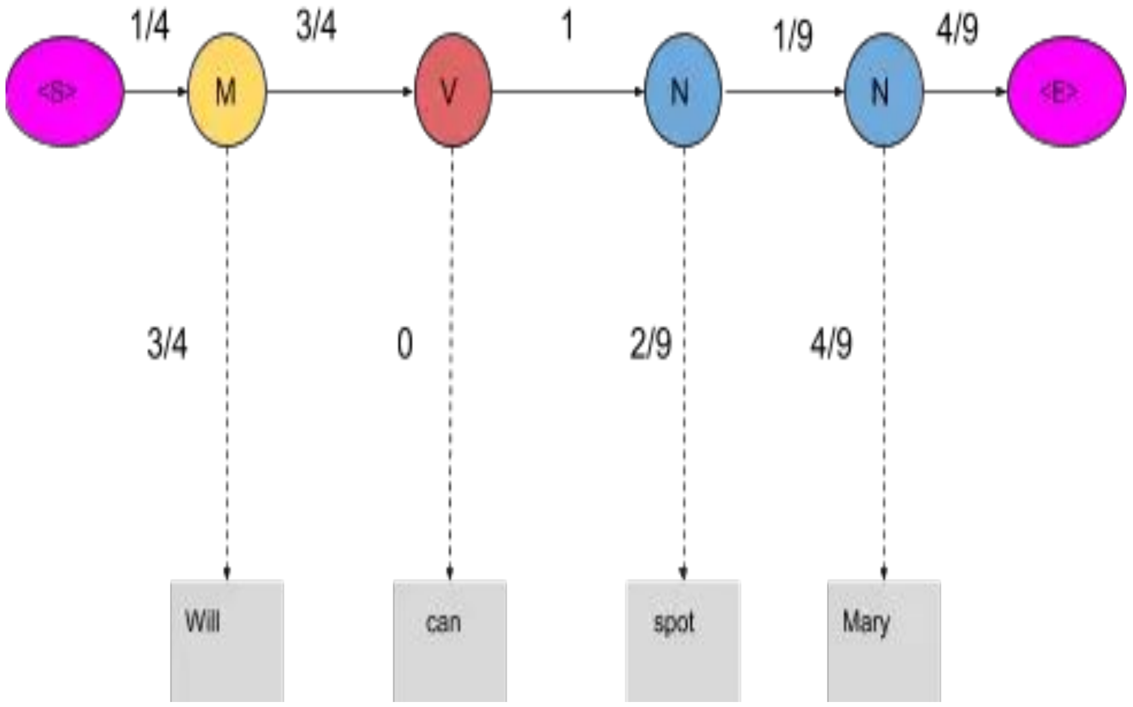
  Will as a model

  Can as a verb

  Spot as a noun

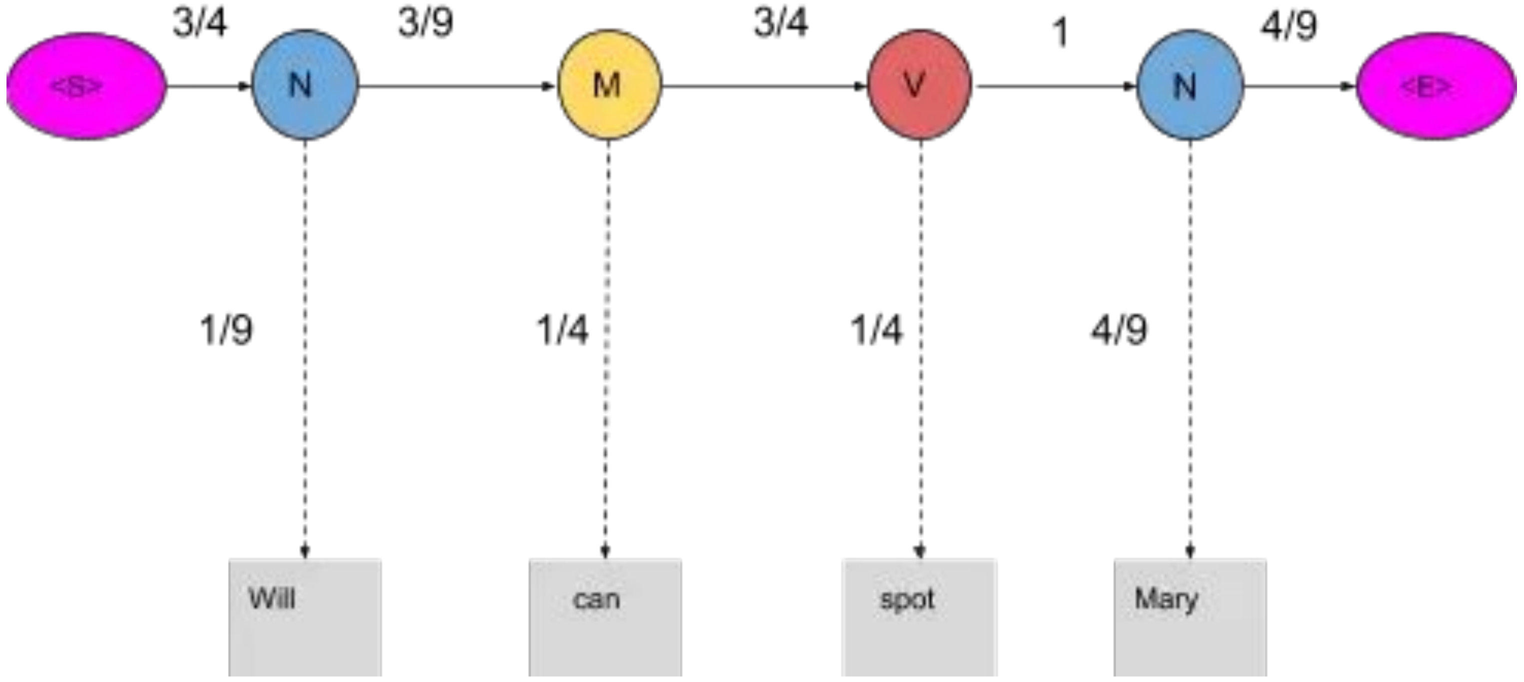
  Mary as a noun
- Now calculate the probability of this sequence being correct in the following manner.
- The probability of the tag Model (M) comes after the tag <S> is ¼ as seen in the table. Also, the probability that the word Will is a Model is 3/4. In the same manner, we calculate each and every probability in the graph. Now the product of these probabilities is the likelihood that this sequence is right. Since the tags are not correct, the product is zero.

  1/4*3/4*3/4*0*1*2/9*1/9*4/9*4/9=0

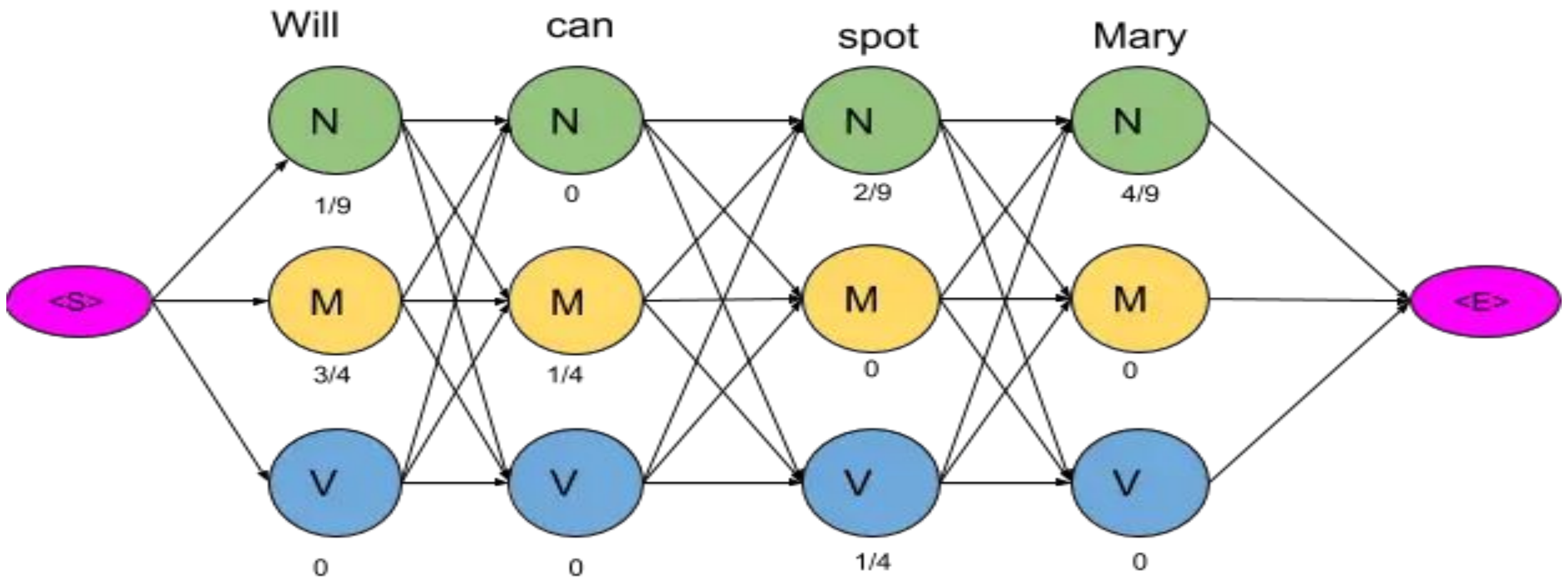


- When these words are correctly tagged, we get a probability greater than zero as shown below
- Calculating the product of these terms we get,

3/4*1/9*3/9*1/4*3/4*1/4*1*4/9*4/9=0.00025720164



- For our example, keeping into consideration just three POS tags we have mentioned, 81 different combinations of tags can be formed.
- In this case, calculating the probabilities of all 81 combinations seems achievable. But when the task is to tag a larger sentence and all the POS tags in the Penn Treebank project are taken into consideration, the number of possible combinations grows exponentially and this task seems impossible to achieve.
- Now let us visualize these 81 combinations as paths and using the transition and emission probability mark each vertex and edge as shown below.

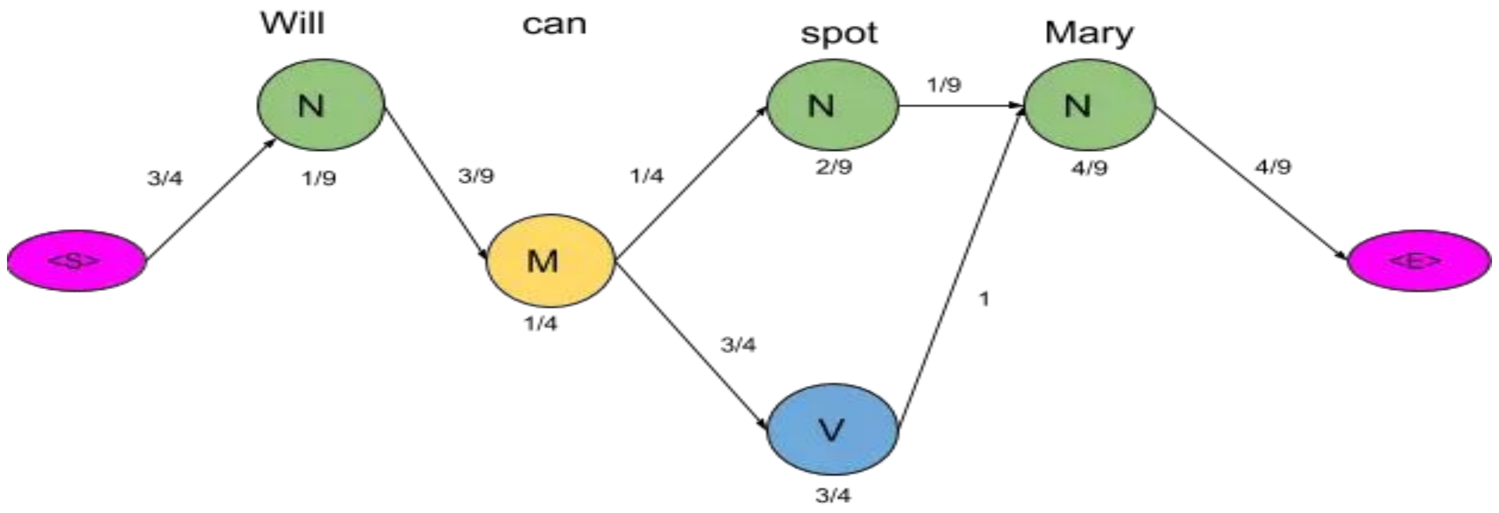


- The next step is to delete all the vertices and edges with probability zero, also the vertices which do not lead to the endpoint are removed. Also, we will mention-
- Now there are only two paths that lead to the end, let us calculate the probability associated with each path.

<S>→N→M→N→N→<E> =3/4*1/9*3/9*1/4*1/4*2/9*1/9*4/9*4/9=0.00000846754

<S>→N→M→N→V→<E>=3/4*1/9*3/9*1/4*3/4*1/4*1*4/9*4/9=0.00025720164

- Clearly, the probability of the second sequence is much higher and hence the HMM is going to tag each word in the sentence according to this sequence.



Parsing Basic concepts:

Top-Down Parsing

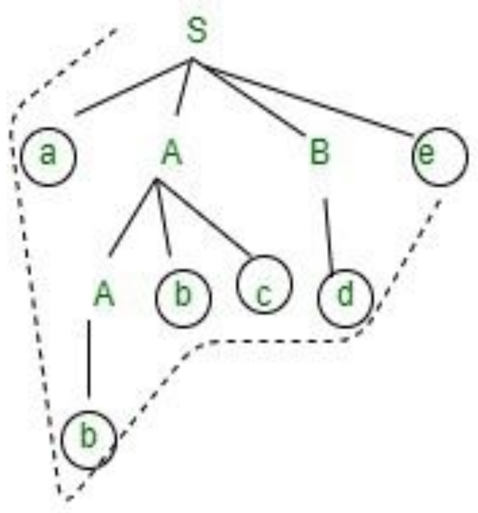
- In top down technique parse tree constructs from top and input will read from left to right. In top down, In top down parser, It will start symbol from proceed to string.
- It follows left most derivation.
- In top down parser, difficulty with top down parser is if variable contain more than one possibility selecting 1 is difficult.

Working of Top Down Parser :

Let’s consider an example where grammar is given and you need to construct a parse tree by using top down parser technique.

Input – **abbcde$**

- Now, you will see that how top down approach works. Here, you will see how you can generate a input string from the grammar for top down approach.
    - First, you can start with S -> a A B e and then you will see input string a in the beginning and e in the end.
    - Now, you need to generate abbcde .
    - Expand A-> Abc and Expand B-> d.
    - Now, You have string like aAbcde and your input string is abbcde.
    - Expand A->b.
    - Final string, you will get abbcde.
- Given diagram is explanation for constructing top down parse tree. You can see clearly in the diagram how you can generate the input string using grammar with top down approach.



Bottom-Up Parsing

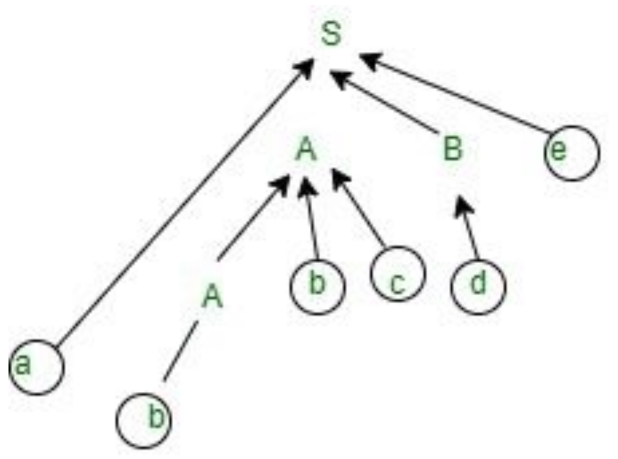
- In Bottom-up the derivation will start from string and proceed to start.
- Here identifying the correct handle (substring) is always difficult.
- It will follow rightmost derivation in reverse order.
- Working of Bottom-up parser :
- Let’s consider an example where grammar is given and you need to construct a parse tree by using bottom-up parser technique.
- Example – S -> aABe

  A -> Abc | b

  B -> d
- Now, let’s consider the input to read and to construct a parse tree with bottom-up approach.

Input – abbcde$

- Now, you will see that how bottom-up approach works. Here, you will see how you can generate a input string from the grammar for bottom-up approach.
    - First, you can start with A -> b.
    - Now, expand A -> Abc.
    - After that Expand B-> d.
    - In the last, just expand the S -> aABe
    - Final string, you will get abbcde.



- Given below is the Diagram explanation for constructing bottom-up parse tree. You can see clearly in the diagram how you can generate the input string using grammar with bottom-up approach.

Treebank Parsing

- Treebank parsing is a method used to analyze and understand the syntactic structure of sentences using annotated corpora called treebanks.
- Treebank parsing involves analyzing a sentence’s syntactic structure by utilizing a treebank, which is a large, annotated collection of text. Each sentence in a treebank is annotated with a parse tree that illustrates its grammatical structure.

Components of Treebank Parsing

a. Treebank: A treebank is a linguistic resource where sentences are annotated with their syntactic structures in the form of trees. It provides a corpus for training, evaluating, and developing parsing models and other NLP tools.

b. Parse Tree: A parse tree (or syntax tree) is a hierarchical representation of a sentence’s syntactic structure. It shows how words and phrases are related to each other within the sentence.

Types of Treebanks

a. Phrase Structure Treebanks

Annotate sentences with phrase structure grammars. They depict how sentences are broken down into noun phrases (NP), verb phrases (VP), etc.

Example: Penn Treebank.

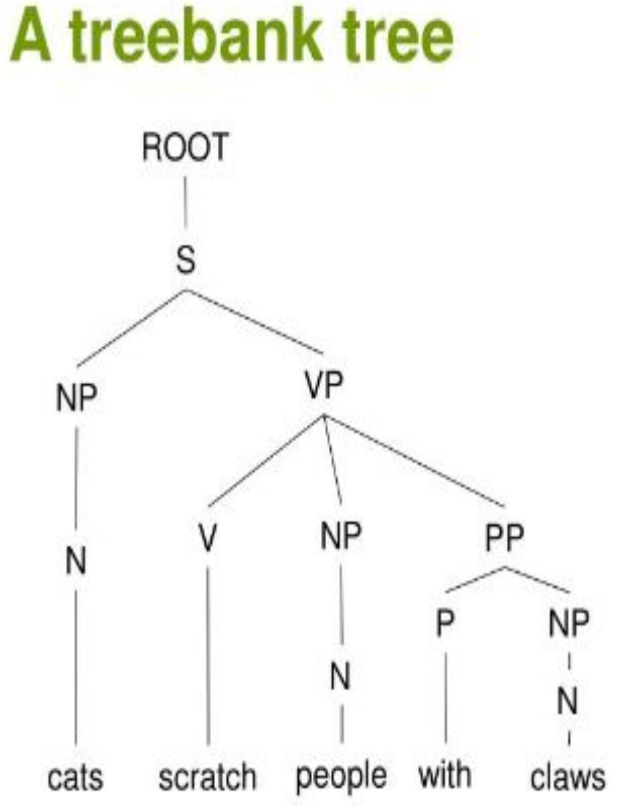
Annotation: Includes labels like S (sentence), NP (noun phrase), VP (verb phrase), PP (prepositional phrase), etc.

b. Dependency Treebanks

Annotate sentences with dependency structures, focusing on the relationships between words rather than phrase structures.

Example: Universal Dependencies.

Annotation: Includes dependency relations such as subject, object, and modifier.



Parsing Techniques

a. Rule-Based Parsing: Uses a set of predefined grammatical rules to analyze sentences.

Example: Context-Free Grammar (CFG) based parsers.

Process: Apply grammatical rules to construct parse trees.

b. Statistical Parsing: Uses probabilistic models trained on treebank data to predict the most likely parse tree for a sentence.

Example: Probabilistic Context-Free Grammar (PCFG), Hidden Markov Models (HMM).

Process: Train models on annotated treebank data to estimate probabilities of different parse structures.

c. Dependency Parsing: Focuses on identifying the syntactic relationships between words in a sentence.

Example: Dependency grammars, such as those used in the Universal Dependencies project.

Process: Construct a dependency tree where nodes represent words, and edges represent syntactic relations.

Syntactic parsing: CKY parsing

- The CKY (Cocke-Kasami-Younger) parsing algorithm is a fundamental technique used in computational linguistics for syntactic parsing.
- It is designed to parse sentences based on context-free grammars (CFGs) that have been converted into Chomsky Normal Form (CNF).
- The algorithm uses dynamic programming to efficiently determine if a given sentence can be generated by a grammar and to construct the corresponding parse tree.

Chomsky Normal Form (CNF)

- A context-free grammar is in Chomsky Normal Form if all its production rules are of the form:

  $A \rightarrow BC$

  $A \rightarrow a$

  $S \rightarrow \epsilon$ (if the language includes the empty string)

- Where:
  - A,B,C are non-terminal symbols.
  - a is a terminal symbol.
  - S is the start symbol.
  - ϵ is the empty string.

Steps in CKY Parsing

1. Grammar Conversion to CNF: Convert the given CFG into CNF. This is necessary because the CKY algorithm requires all productions to be binary or unary.
2. Initialization: Create a table (a 2D array) where the cell table[i][j] will store all the non-terminals that can generate the substring from position i to j in the sentence.
3. Filling the Table:
   a. For each length of substring (starting from 1 up to the length of the sentence):
   b. For each starting position in the sentence:
   c. For each possible split of the substring:
   d. Check if there are non-terminals in the table that can generate the two halves of the substring.
   e. If there are, check the grammar rules to see if there is a non-terminal that can combine these two halves.
   f. Add this non-terminal to the current cell in the table.
4. Backtracking to Build Parse Tree: If the start symbol is found in the top-right cell of the table, the sentence can be generated by the grammar. Backtracking is then used to construct the parse tree from the table.

Consider a sample grammar in Chomsky Normal Form:

NP --> Det | Nom

Nom --> AP | Nom

AP --> Adv | A

Det --> a | an

Adv --> very | extremely

AP --> heavy | orange | tall

A --> heavy | orange | tall | muscular

Nom --> book | orange | man

Now consider the phrase, “a very heavy orange book“

Let us start filling up the table from left to right and bottom to top, according to the rules described above:

Step 1: Create a table with rows and columns as the words in the sentence given above

Step 2: Enter the values of the similar words intersecting each other on the table with their production as follows:

T[1][1] = Det

T[2][2] = Adv

T[3][3] = AP | A

T[4][4] = A | AP | Nom

T[5][5] = Nom

Step 3: Calculate the values for empty columns as follows:

T[1][2] = T[1][1] T[2][2]

T[1][2] = Det | Adv = □

(Since no such production is present in the grammar)

T[2][3] = T[2][2] T[3][3]

T[2][3] = Adv|AP or Adv|A

T[2][3] = □ or **AP**

T[1][3] = T[1][2] T[3][3]

T[1][3] = □ AP = □

OR

T[1][3] = T[1][1] T[2][3]

T[1][3] = Det | Adv = □

T[3][4] = T[3][3] T[4][4]

T[3][4] = A|Nom or A|A or A|AP

AP|Nom or AP|A or AP|AP

T[3][4] = Nom

T[2][4] = T[2][3] T[4][4]

T[2][4] = AP|Nom or AP|A or AP|AP

T[2][4]= **Nom** or □ or □

OR

T[2][4] = T[2][2] T[3][4]

T[2][4] = □|Nom = □

T[1][4] = T[1][3] T[4][4]

T[1][4] = □|AP or □|A or □| Nom = □

OR

T[1][4] = T[1][2] T[3][4]

T[1][4] = □|AP or □|A or □| Nom = □

OR

T[1][4] = T[1][1] T[2][4]

T[1][4] = Det | Nom

T[1][4] = NP

Similarly we find value for

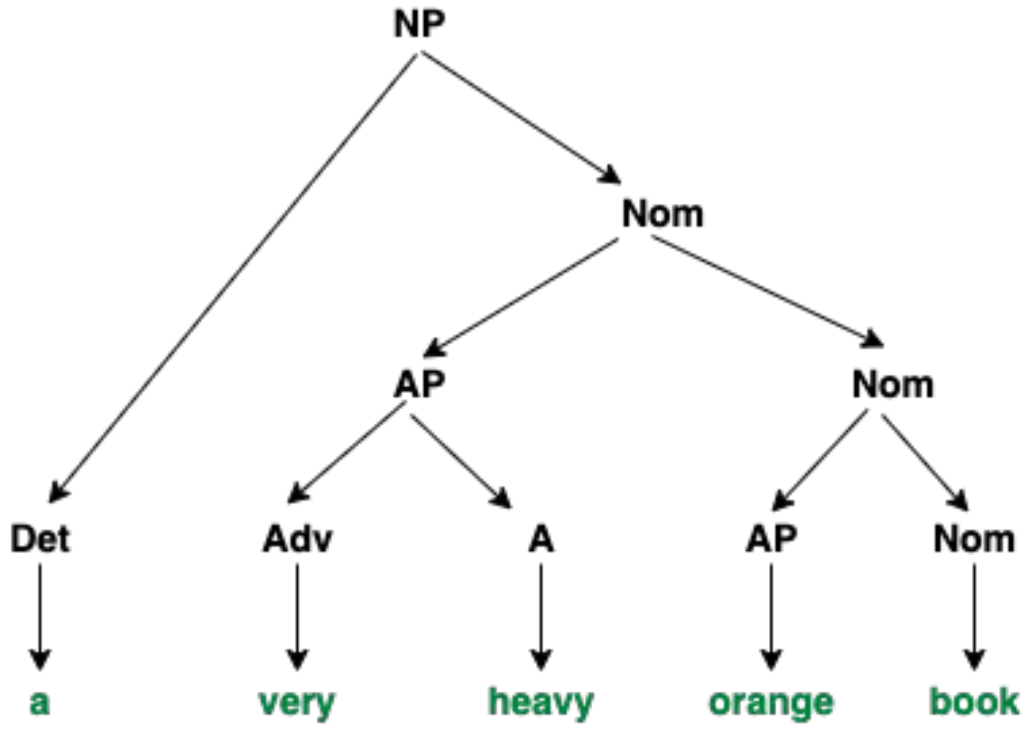
T[4][5] = Nom

T[3][5] = Nom

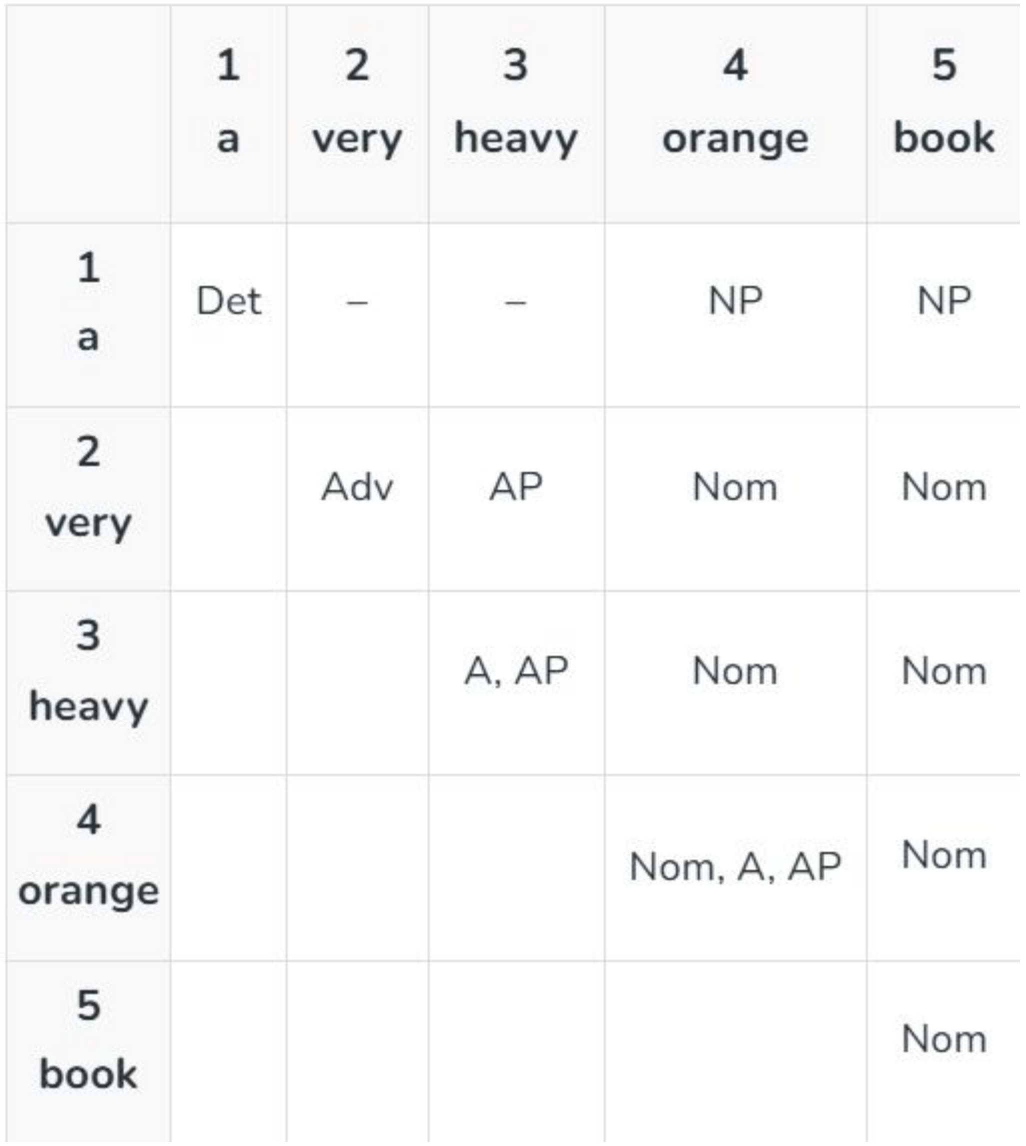
T[2][5] = Nom

T[1][5] = NP

The parse tree of this phrase would look like this:



Let us look at another example phrase, “a very tall extremely muscular man”



	1 a	2 very	3 heavy	4 orange	5 book
1 a	Det	–	–	NP	NP
2 very		Adv	AP	Nom	Nom
3 heavy			A, AP	Nom	Nom
4 orange				Nom, A, AP	Nom
5 book					Nom

Statistical Parsing basics: Probabilistic Context-Free Grammar (PCFG)

Probabilistic Context-Free Grammars (PCFGs) are an extension of Context-Free Grammars (CFGs) that assign probabilities to each production rule. This allows the grammar to not only determine if a sentence can be generated by the grammar but also to find the most probable parse tree for a sentence. PCFGs are widely used in natural language processing for tasks like syntactic parsing.

PCFG consists of the following components:

1) A set of non-terminal symbols (N): These are abstract symbols that can be expanded into other non-terminal or terminal symbols.
2) A set of terminal symbols (Σ): These are the actual symbols of the language (words in the case of natural language).
3) A set of production rules (P): Rules for expanding non-terminal symbols, each associated with a probability (e.g., S -> NP VP [1.0]).
4) A start symbol (S): The non-terminal symbol from which parsing starts.

Example of a PCFG

Here's an example of a simple PCFG for a fragment of English:

Properties

Rule Probabilities: The probabilities of all rules expanding a given non-terminal must sum to 1.

Parse Trees: A sentence can have multiple parse trees, and the probability of a parse tree is the product of the probabilities of the rules used in that tree.

Example: From the above grammar find the probability of the sentence given: “John saw the man with the telescope”

For each word, identify its corresponding terminal rules.

"John" -> NP [0.2]

"saw" -> V [0.5]

"the" -> Det [0.5]

"man" -> N [0.3]

"with" -> P [0.5]

"telescope" -> N [0.2]

The total probability of this parse tree is:

P(S -> NP VP) * P(NP -> 'John') * P(VP -> V NP PP) * P(V -> 'saw') *

P(NP -> Det N) * P(Det -> 'the') * P(N -> 'man') * P(PP -> P NP) *

P(P -> 'with') * P(NP -> Det N) * P(Det -> 'the') * P(N -> 'telescope')

1.0 * 0.2 * 0.5 * 0.5 * 0.4 * 0.5 * 0.3 * 1.0 * 0.5 * 0.4 * 0.5 * 0.2 = 0.000036

"John" -> NP [0.2]

"saw" -> V [0.5]

"the" -> Det [0.5]

"man" -> N [0.3]

"with" -> P [0.5]

"telescope" -> N [0.2]

The total probability of this parse tree is:

P(S -> NP VP) * P(NP -> 'John') * P(VP -> V NP PP) * P(V -> 'saw') *

P(NP -> Det N) * P(Det -> 'the') * P(N -> 'man') * P(PP -> P NP) *

P(P -> 'with') * P(NP -> Det N) * P(Det -> 'the') * P(N -> 'telescope')

1.0 * 0.2 * 0.5 * 0.5 * 0.4 * 0.5 * 0.3 * 1.0 * 0.5 * 0.4 * 0.5 * 0.2 = 0.000036

S -> NP VP [1.0]

VP -> V NP [0.5] | V NP PP [0.5]

PP -> P NP [1.0]

V -> 'saw' [0.5] | 'ate' [0.3] | 'walked' [0.2]

NP -> 'John' [0.2] | 'Mary' [0.2] | 'Bob' [0.2] | Det N [0.2] | Det N PP [0.2]

Det -> 'a' [0.5] | 'the' [0.5]

N -> 'man' [0.2] | 'dog' [0.2] | 'cat' [0.2] | 'telescope' [0.2] | 'park' [0.2]

P -> 'in' [0.25] | 'on' [0.25] | 'by' [0.25] | 'with' [0.25]

Statistical Parsing basics: Probabilistic CKY Parsing of PCFGs

Probabilistic CKY (Cocke-Younger-Kasami) parsing is a dynamic programming algorithm used to find the most probable parse tree for a given sentence under a Probabilistic Context-Free Grammar (PCFG). It extends the standard CKY parsing algorithm by incorporating probabilities associated with production rules.

Steps of Probabilistic CKY Parsing

1. Initialization: Initialize a 3D table where each cell stores the maximum probability of a non-terminal producing a substring of the input sentence.
2. Filling the Table: For each substring of the input sentence, compute the maximum probability of each non-terminal producing that substring using the production rules and their probabilities.
3. Backtracking: Construct the most probable parse tree by backtracking through the table.