

**Understanding TF-IDF with example**

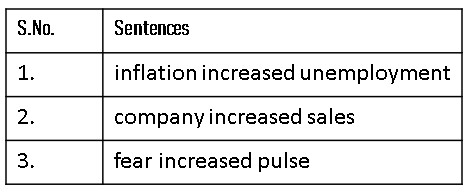
Let’s understand the working of TF-IDF with the example. Assume we have three sentences given below:

1. Inflation has increased unemployment
2. The company has increased its sales
3. Fear increased his pulse

TF-IDF can be implemented in four steps for representing the above 3 sentences.

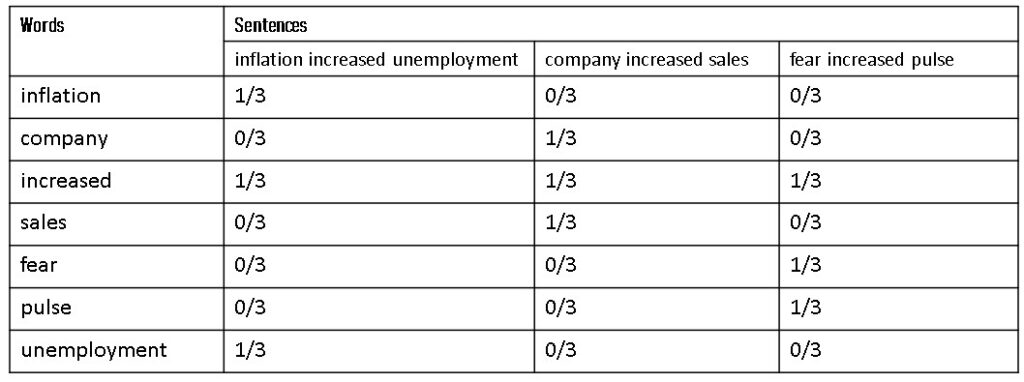
**Step 1: Data Pre-processing**

After lowercasing and removing stop words the sentences are transformed as below:

**Sentences after data pre-processing**

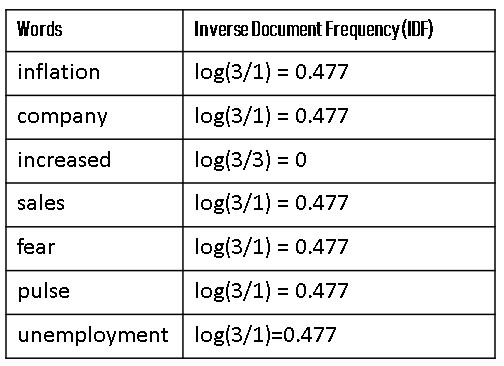
**Step 2: Calculating Term Frequency**

In this step, we have to calculate TF i.e., the **Term Frequency** of our given sentences.

The term frequency of each of the words in the sentences

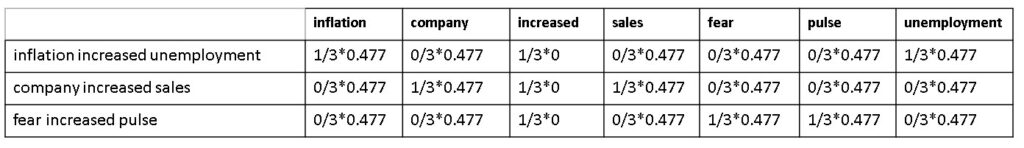
**Step 3: Calculating Inverse Document Frequency**

Now, next, we have to calculate the **Inverse Document Frequency** **(IDF)** of all the words in the sentences.

Inverse document frequency of all the words in sentences

**Step 4: Calculating Product of Term Frequency** **& Inverse Document Frequency**

Now, the last step is to take the product of the term frequency of each of the words with their inverse document frequency. Now the table of TF-IDF will look like as below:



After simplifying the above table we will get the final TF-IDF matrix as follows:



Count Vectorizer

*document = [ “One Geek helps Two Geeks”, “Two Geeks help Four Geeks”, “Each Geek helps many other Geeks at GeeksforGeeks.”]*

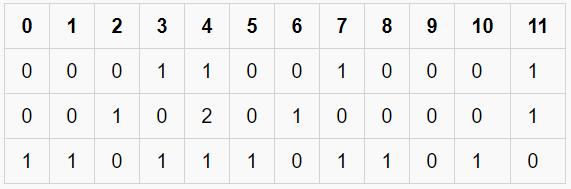
CountVectorizer creates a matrix in which each unique word is represented by a column of the matrix, and each text sample from the document is a row in the matrix. The value of each cell is nothing but the count of the word in that particular text sample.  This can be visualized as follows –

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **at** | **each** | **four** | **geek** | **geeks** | **geeksforgeeks** | **help** | **helps** | **many** | **one** | **other** | **two** |
| document[0] | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| document[1] | 0 | 0 | 1 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| document[2] | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |

**Key Observations:**

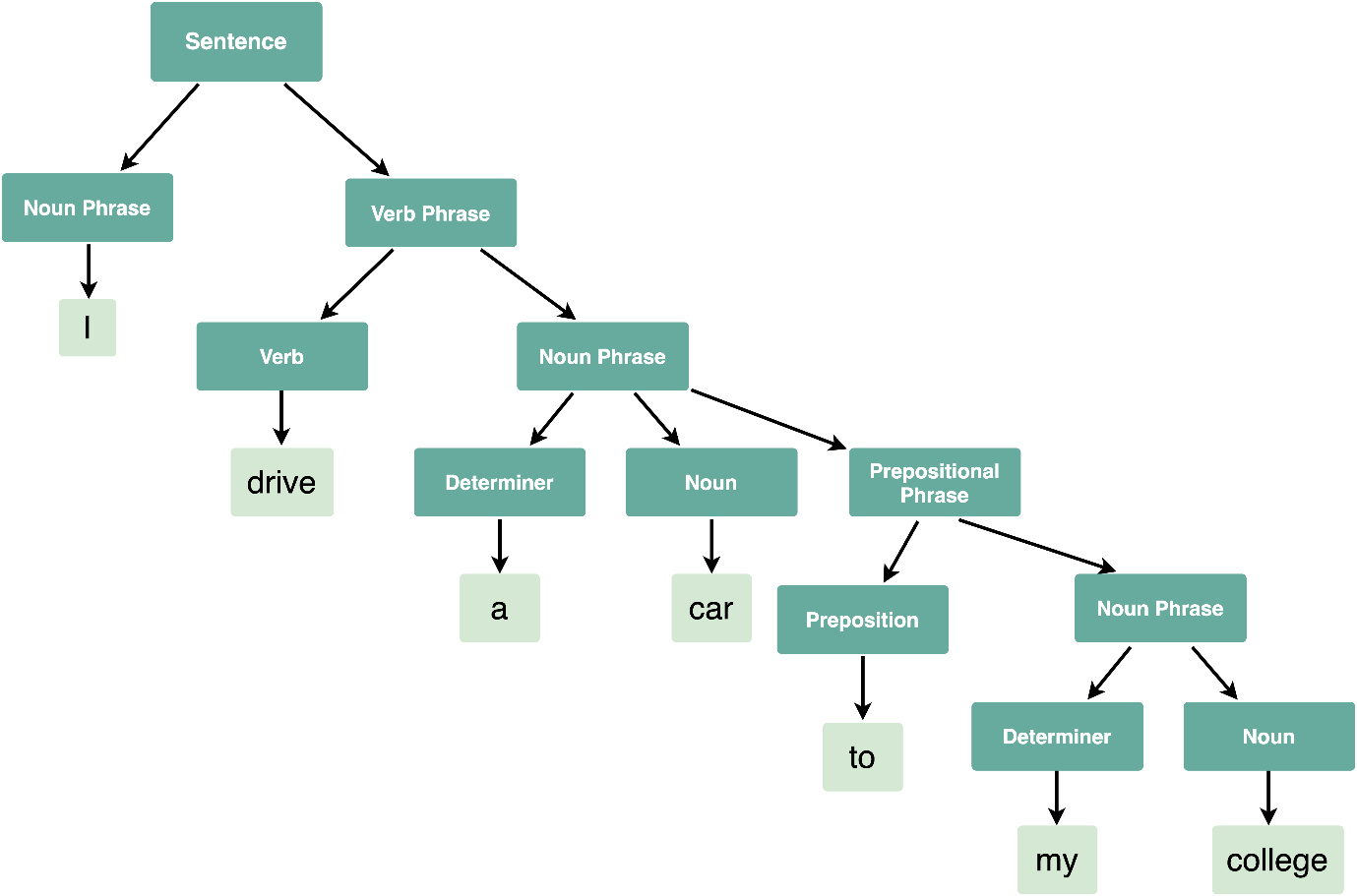
1. There are 12 unique words in the document, represented as columns of the table.
2. There are 3 text samples in the document, each represented as rows of the table.
3. Every cell contains a number, that represents the count of the word in that particular text.
4. All words have been converted to lowercase.
5. The words in columns have been arranged alphabetically.

Inside CountVectorizer, these words are not stored as strings. Rather, they are given a particular index value. In this case, ‘at’ would have index 0, ‘each’ would have index 1, ‘four’ would have index 2 and so on. The actual representation has been shown in the table below –



*A Sparse Matrix*

This way of representation is known as a **Sparse Matrix**. **Code: Python implementation of CountVectorizer**



1. **Sentiment Analysis**:
   * **Explanation**: Sentiment analysis, also known as opinion mining, involves analyzing text to determine the sentiment expressed within it, such as positive, negative, or neutral. This is useful for understanding public opinion, customer feedback, social media monitoring, and brand perception analysis.
   * **Example**: Analyzing product reviews to determine whether customers are satisfied or dissatisfied with a product.
2. **Named Entity Recognition (NER)**:
   * **Explanation**: Named Entity Recognition involves identifying and classifying named entities mentioned in text into predefined categories such as names of persons, organizations, locations, dates, etc. This helps in information extraction and structuring unstructured text data.
   * **Example**: Extracting names of people, organizations, and locations from news articles for data analysis.
3. **Text Summarization**:
   * **Explanation**: Text summarization involves automatically generating a concise and coherent summary of a longer text while preserving its key information. This is useful for quickly understanding the main points of large documents, news articles, or research papers.
   * **Example**: Summarizing news articles to provide readers with brief overviews of current events.
4. **Machine Translation**:
   * **Explanation**: Machine translation involves translating text from one language to another automatically using computational methods. This is useful for breaking down language barriers and facilitating communication across different linguistic communities.
   * **Example**: Google Translate, which translates text between multiple languages in real-time.
5. **Question Answering**:
   * **Explanation**: Question answering systems automatically process natural language questions and provide relevant answers based on a given knowledge base or corpus of text. This is useful for building virtual assistants, information retrieval systems, and educational tools.
   * **Example**: IBM Watson's question answering system, which competes in the Jeopardy! quiz show by understanding and answering questions posed in natural language.

**Levles of NLP**

**Morphological Analysis**:

* **Explanation**: Morphological analysis involves analyzing the structure and form of words to understand their grammatical properties, such as root words, prefixes, suffixes, and inflections.
* **Example**: Breaking down the word "running" into its morphemes: "run" (root) + "ing" (suffix).

1. **Syntactic Analysis**:
   * **Explanation**: Syntactic analysis involves analyzing the grammatical structure of sentences to understand their syntactic relationships. This includes tasks such as part-of-speech tagging, parsing, and identifying syntactic dependencies.
   * **Example**: Identifying the subject, verb, and object in a sentence and analyzing their relationships.
2. **Semantic Analysis**:
   * **Explanation**: Semantic analysis focuses on understanding the meaning of words, phrases, and sentences in context. This includes tasks such as word sense disambiguation, semantic role labeling, and understanding semantic relationships.
   * **Example**: Understanding that "bank" can refer to a financial institution or the side of a river based on context.

**Discourse Analysis**:

* **Explanation**: Discourse analysis is the study of larger units of language beyond individual sentences, such as conversations, speeches, or written texts. It focuses on understanding the structure, coherence, and organization of these larger units, as well as the relationships between their constituent parts.
* Discourse analysis focuses on studying the structure, organization, and patterns of communication in natural language text beyond the level of individual sentences or utterances. Here's how discourse integration applies to NLP:

**Example**: In a conversation between two people, discourse analysis would involve identifying the topics discussed, understanding the flow of conversation, resolving references to individuals or events, and assessing the overall coherence of the dialogue.

**Pragmatic Analysis**:

* Pragmatic analysis considers the context and intentions behind the language used in communication.
* This phase involves tasks such as discourse analysis, which analyzes the structure and flow of conversations or texts.
* It also includes tasks such as speech act recognition, which identifies the intended actions or purposes of utterances.

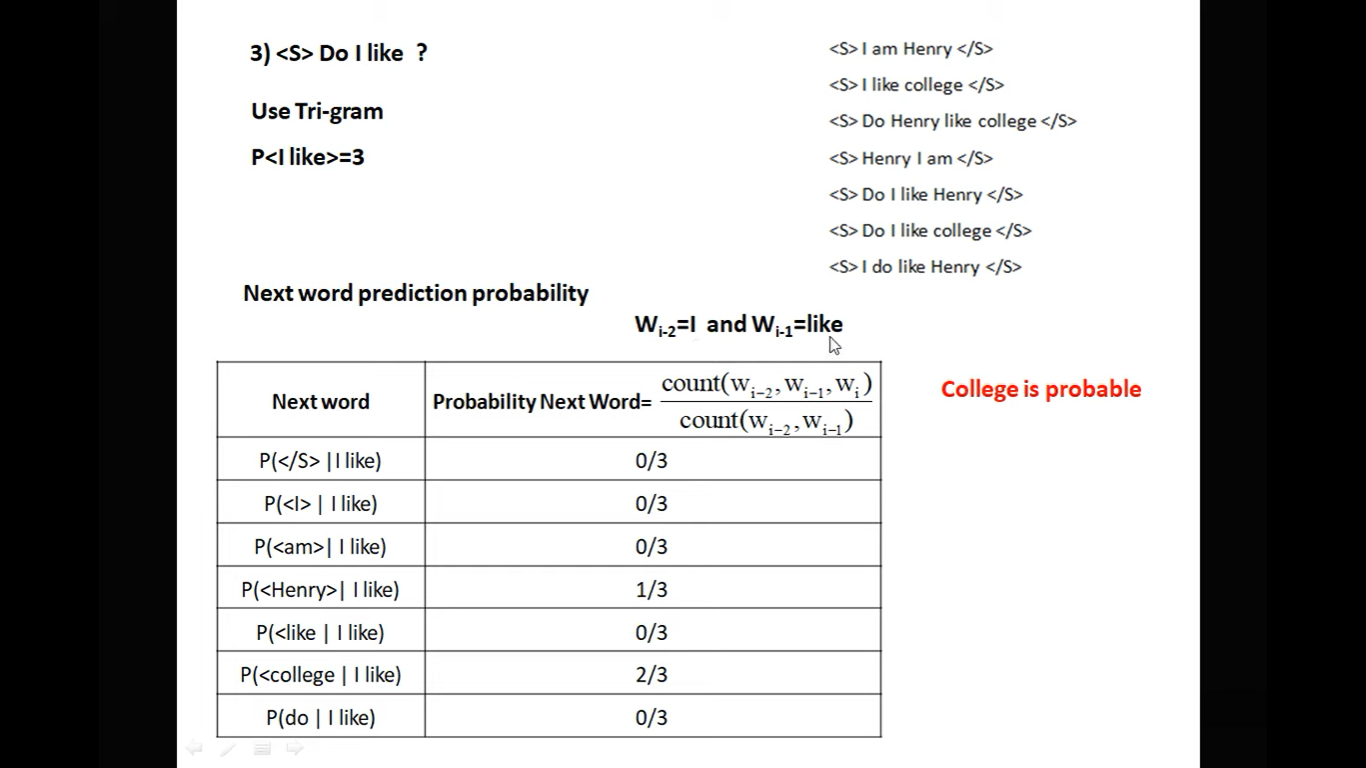
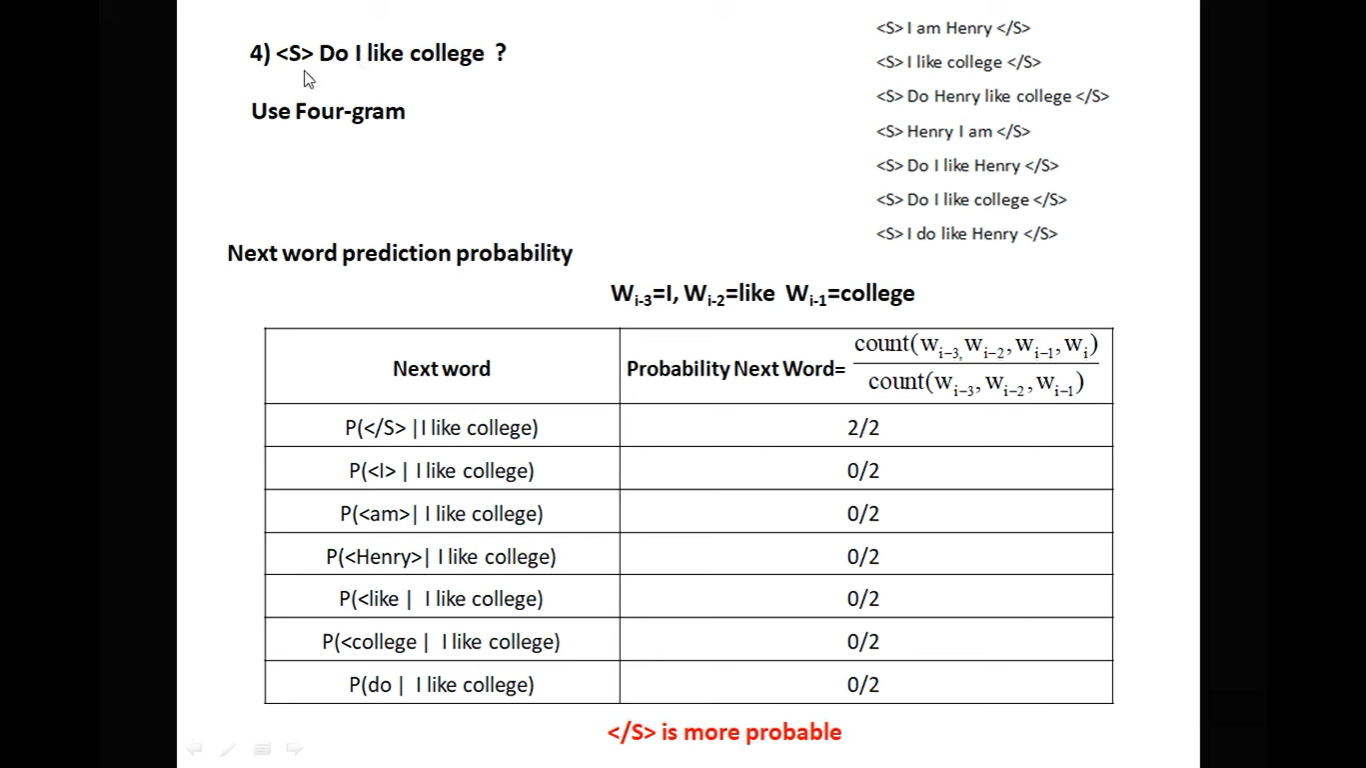
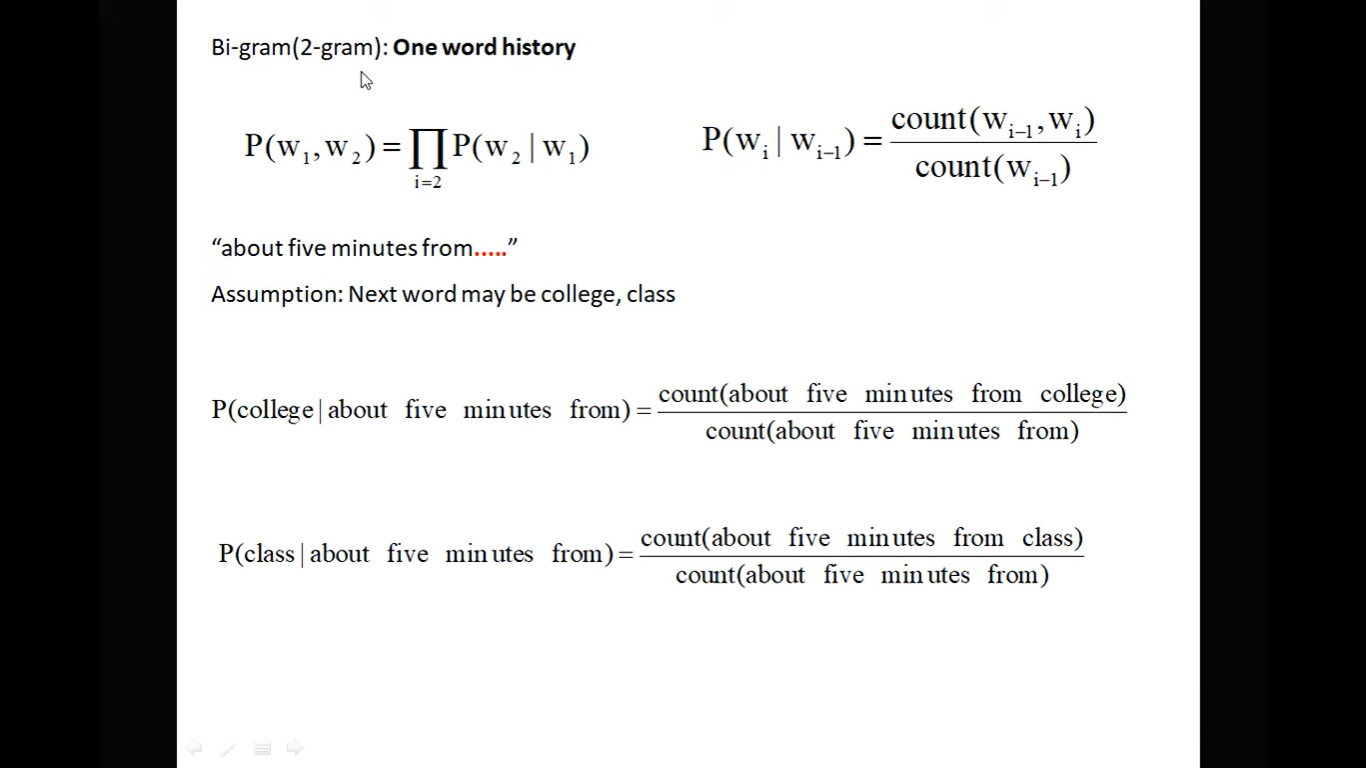
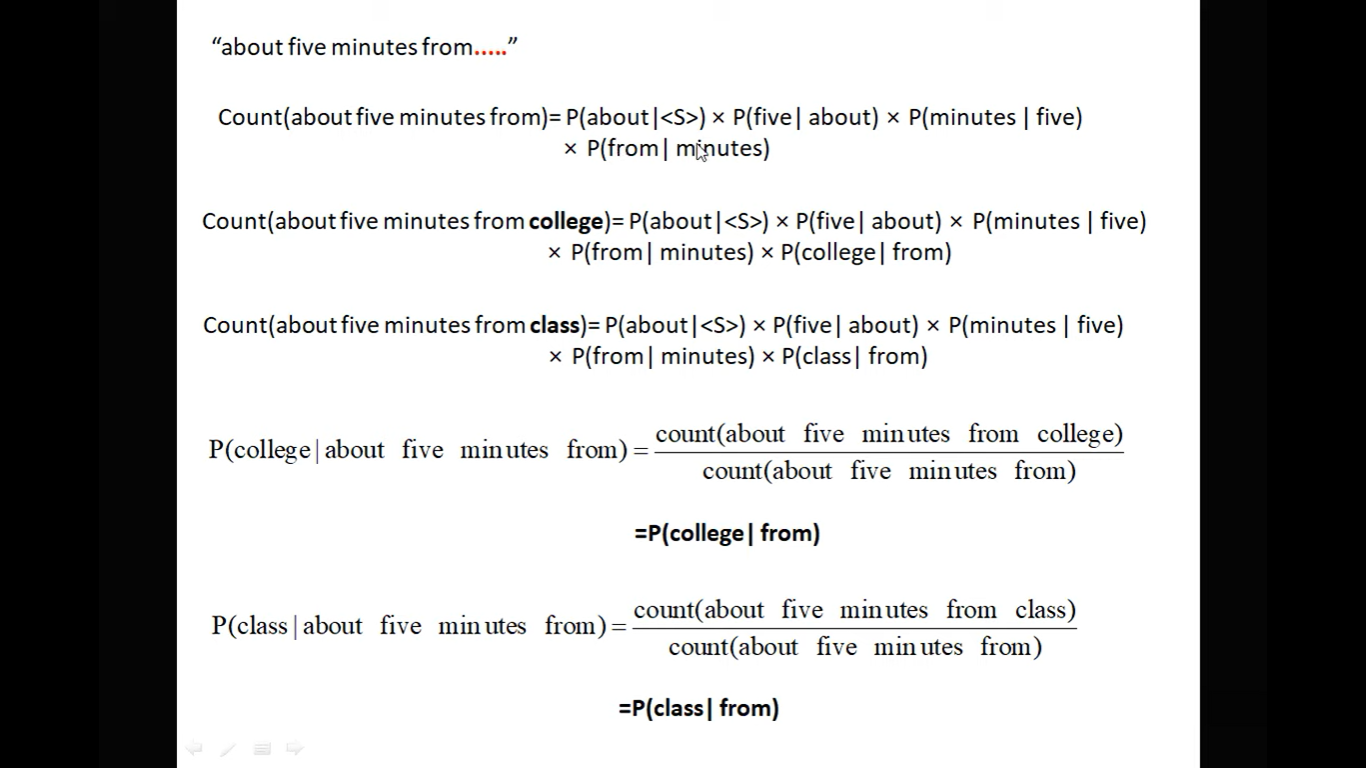
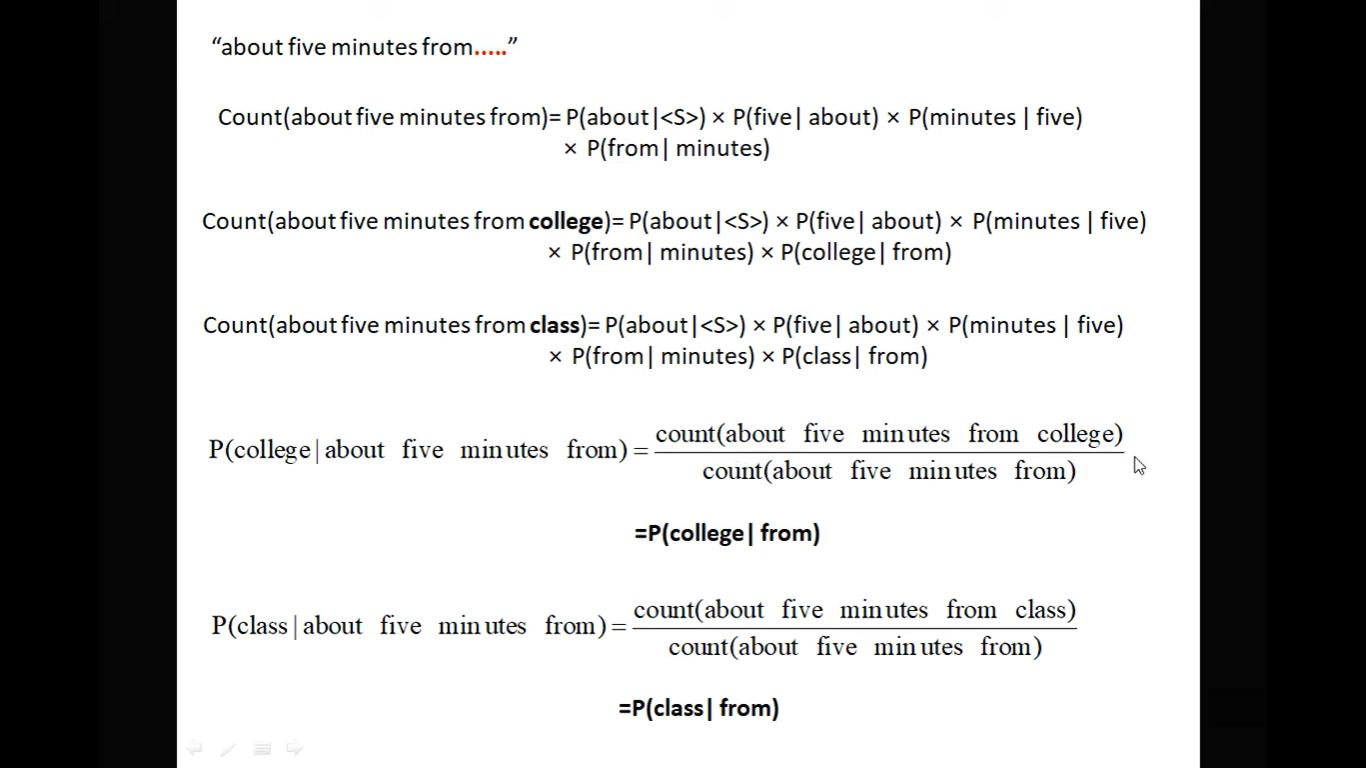
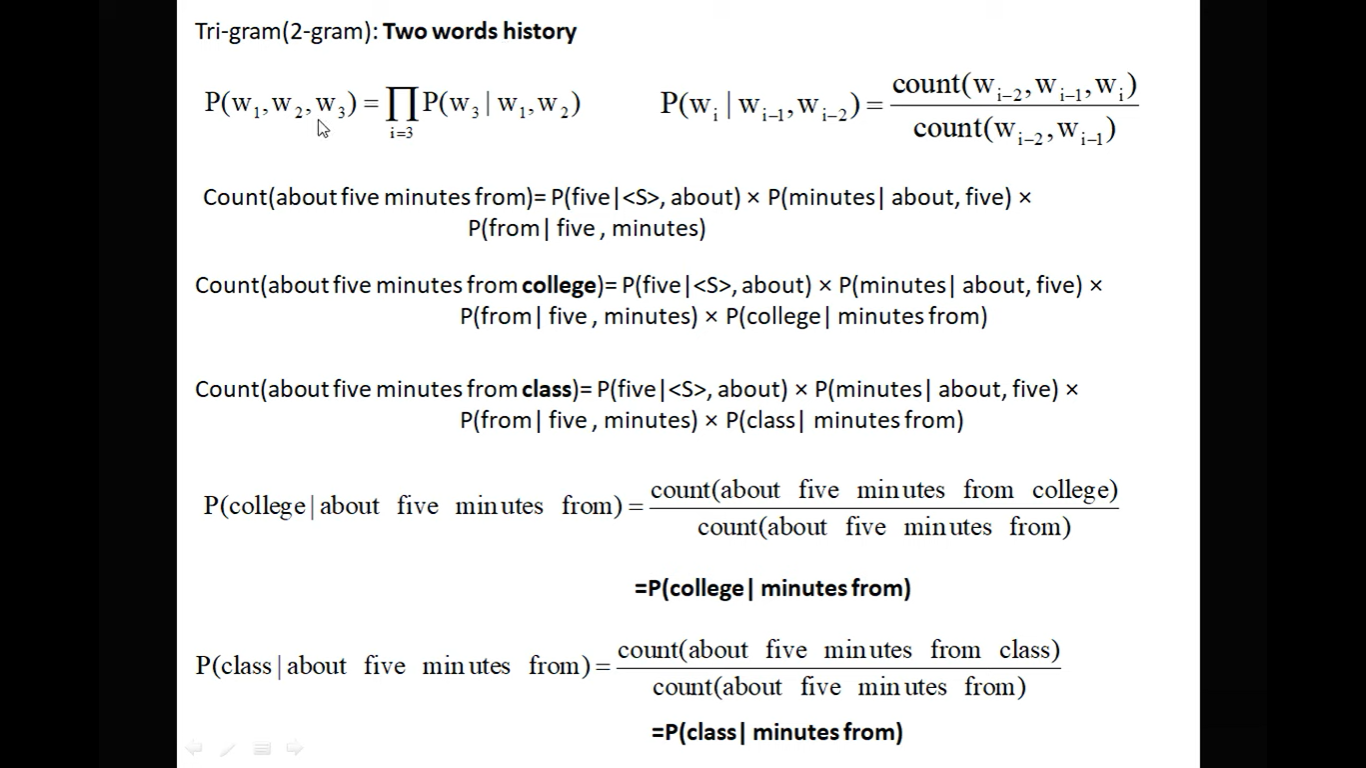
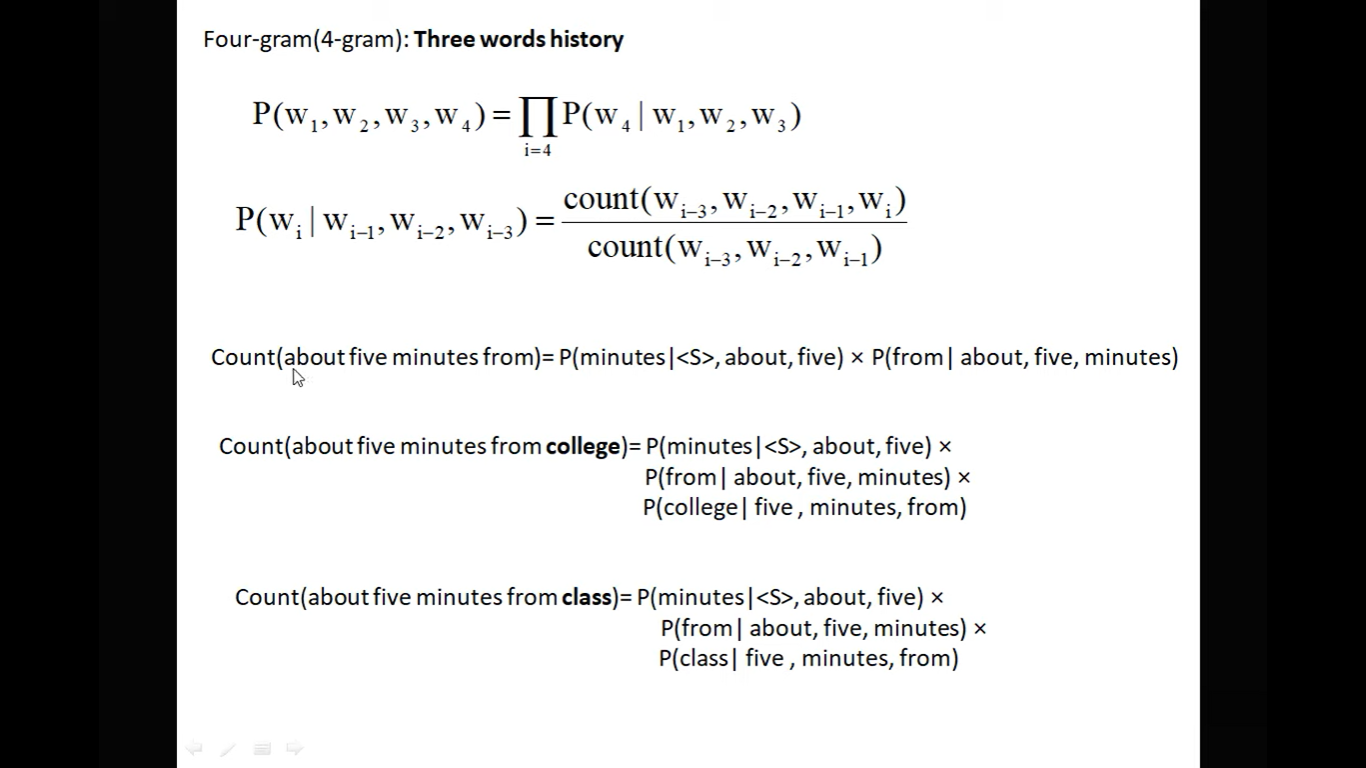
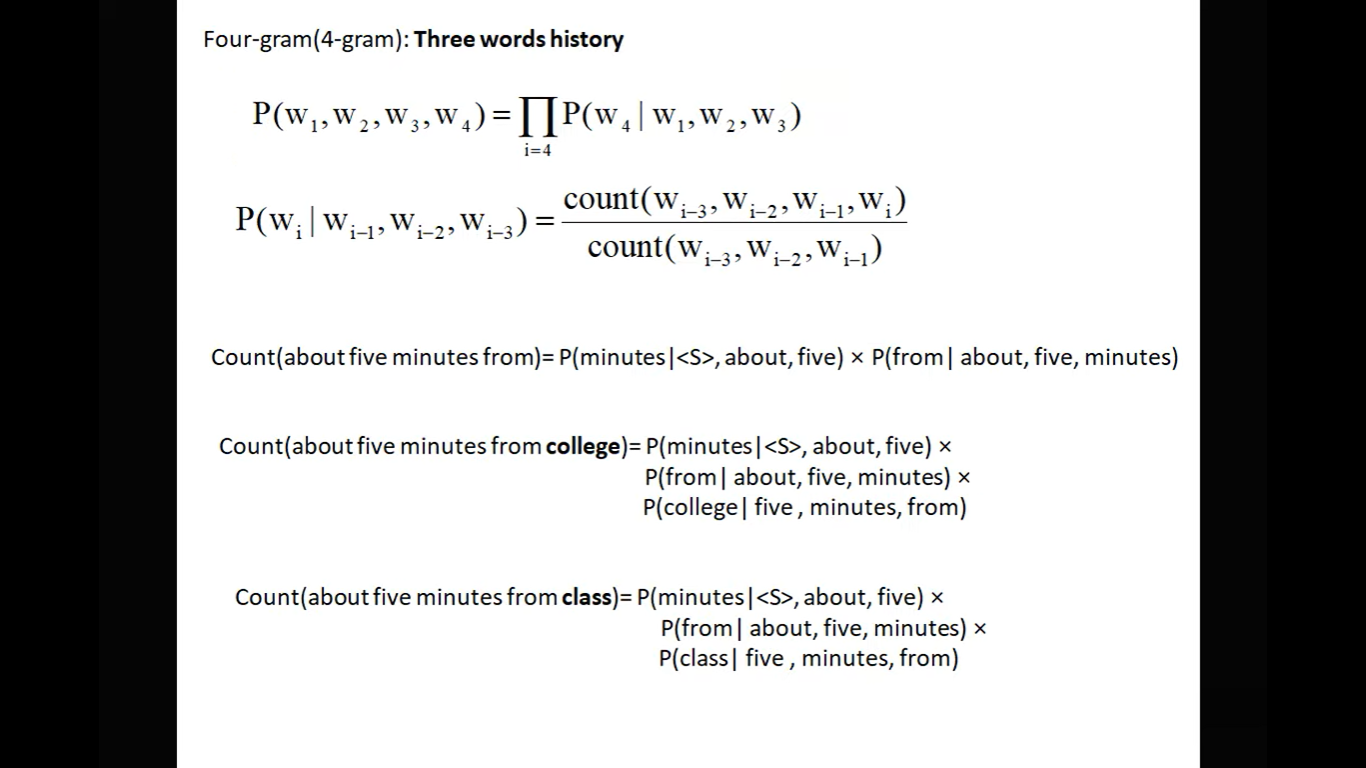
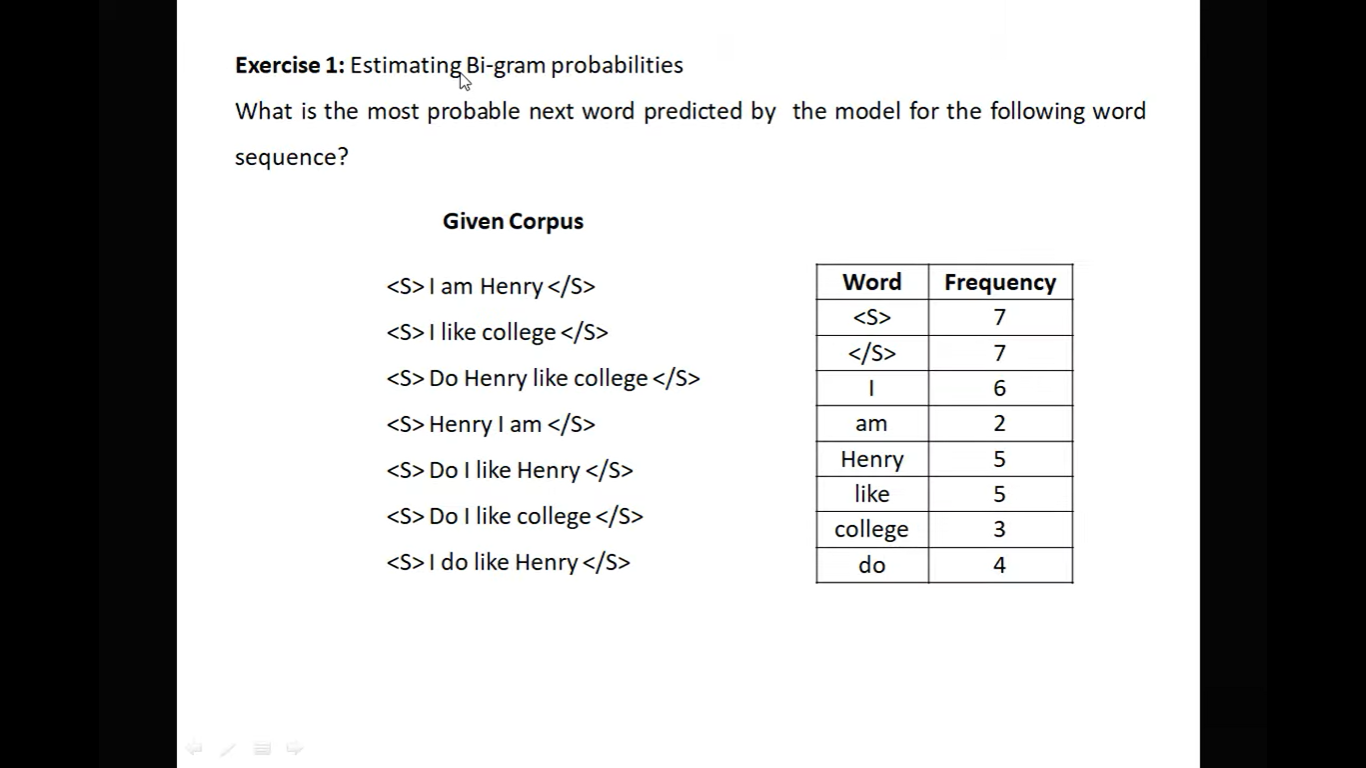
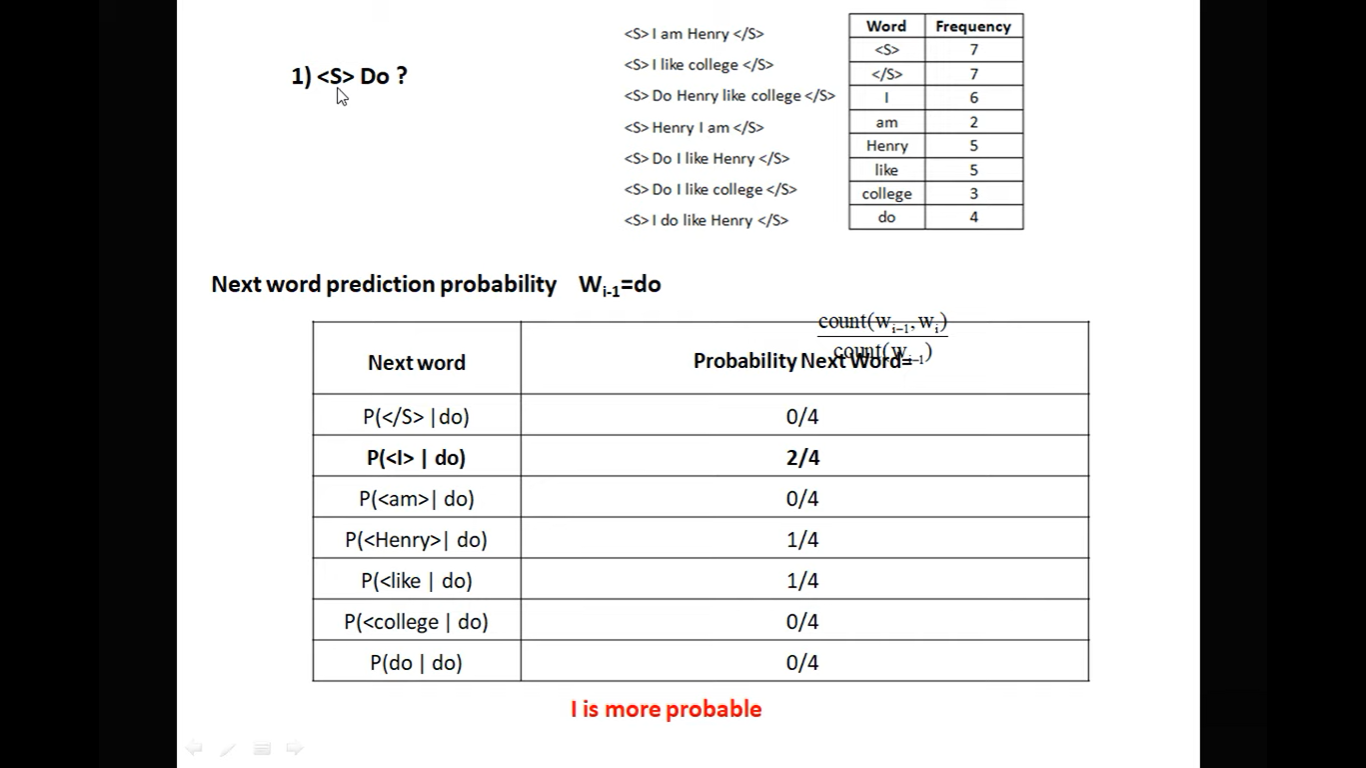
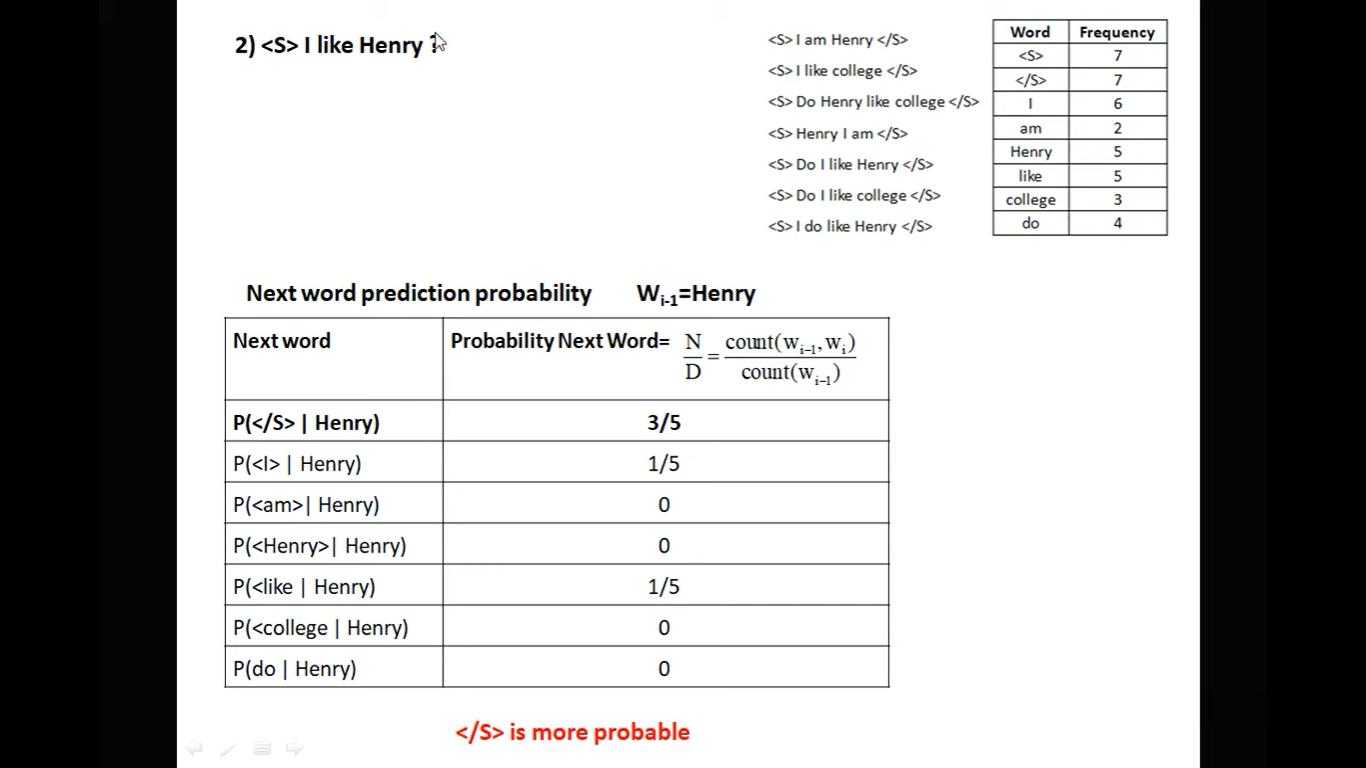
**Example**: When someone says "It's getting late," in a social context, pragmatic analysis involves understanding whether it's a polite hint to end the conversation, an observation about the time, or a request to leave, based on the context and the speaker's intentions.

The CKY algorithm, also known as Cocke-Kasami-Younger algorithm, is a dynamic programming algorithm used for parsing sentences in context-free grammars (CFGs). It efficiently finds all possible parse trees for a given input string and grammar. The CKY algorithm operates by filling up a dynamic programming table in a bottom-up manner, combining smaller substructures to build larger ones until it reaches the top-level structure representing the entire sentence.

What is Word Sense Disambiguation?

Word Sense Disambiguation is an important method of NLP by which the meaning of a word is determined, which is used in a particular context. NLP systems often face the challenge of properly identifying words, and determining the specific usage of a word in a particular sentence has many applications.

Word Sense Disambiguation basically solves the ambiguity that arises in determining the meaning of the same word used in different situations.

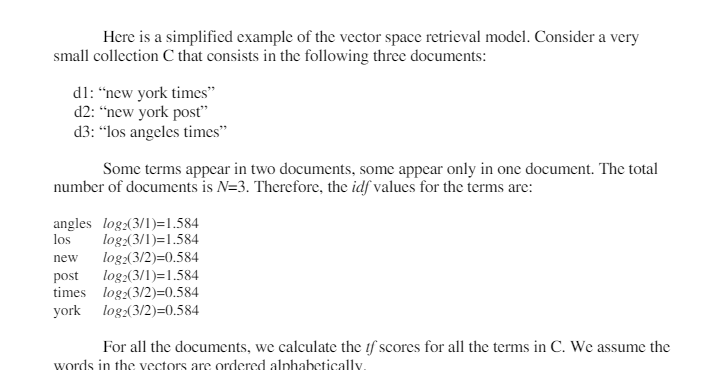
 

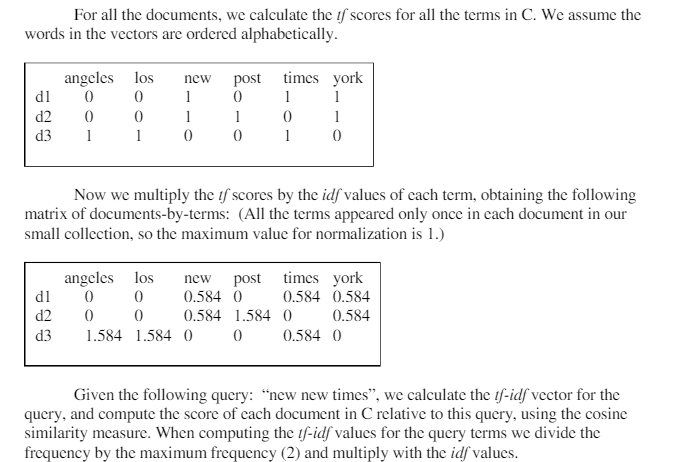
**When to use Jaccard Similarity and when to use Cosine Similarity**:

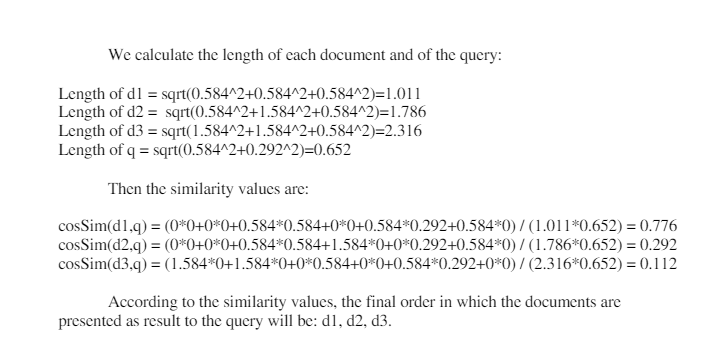
* Jaccard similarity is used when we want to compare the similarity between two sets. It is suitable for binary data or categorical data.
* Cosine similarity, on the other hand, is used when we want to compare the similarity between two vectors in a multidimensional space. It is commonly used for text similarity, recommendation systems, and clustering.

e) **Jaccard Distance**:

* Jaccard distance is a measure of dissimilarity between two sets. It is calculated as 1 minus the Jaccard similarity.
* Jaccard distance=1−Jaccard similarityJaccard distance=1−Jaccard similarity







1. The quick brown fox jumps over the lazy dog."

**Output:**

* + Stemming: "the quick brown fox jump over the lazi dog."
  + Lemmatization: "the quick brown fox jump over the lazy dog."
  + Stop-words Removal: "quick brown fox jumps lazy dog."
  + Lowercase: "the quick brown fox jumps over the lazy dog."
  + POS Tagging:
    - "The/DT quick/JJ brown/JJ fox/NN jumps/VBZ over/IN the/DT lazy/JJ dog/NN."

1. **Text:** "She sells seashells by the seashore."

**Output:**

* + Stemming: "she sell seashel by the seashor."
  + Lemmatization: "she sell seashell by the seashore."
  + Stop-words Removal: "sells seashells seashore."
  + Lowercase: "she sells seashells by the seashore."
  + POS Tagging:
    - "She/PRP sells/VBZ seashells/NNS by/IN the/DT seashore/NN."

1. **Text:** "I am reading a book written by J.K. Rowling."

**Output:**

* + Stemming: "I am read a book written by j.k. rowl."
  + Lemmatization: "I am reading a book write by J.K. Rowling."
  + Stop-words Removal: "reading book written J.K. Rowling."
  + Lowercase: "i am reading a book written by j.k. rowling."
  + POS Tagging:
    - "I/PRP am/VBP reading/VBG a/DT book/NN written/VBN by/IN J.K./NNP Rowling/NNP."

1. **Text:** "Cats are cute animals."

**Output:**

* + Stemming: "cat are cute anim."
  + Lemmatization: "cat be cute animal."
  + Stop-words Removal: "cats cute animals."
  + Lowercase: "cats are cute animals."
  + POS Tagging:
    - "Cats/NNS are/VBP cute/JJ animals/NNS."

1. **Text:** "The sun sets in the west."

**Output:**

* + Stemming: "the sun set in the west."
  + Lemmatization: "the sun set in the west."
  + Stop-words Removal: "sun sets west."
  + Lowercase: "the sun sets in the west."
  + POS Tagging:
    - "The/DT sun/NN sets/VBZ in/IN the/DT west/NN."

**Ambiguity Sentences:**

"I saw the man with the telescope."

**Parse Trees:**

* "I saw (the man) (with the telescope)." (I used the telescope to see the man.)
* "I saw (the man with the telescope)." (The man had the telescope.)

"The chicken is ready to eat."

**Parse Trees:**

* "The (chicken) (is (ready) (to eat))." (The chicken is prepared for consumption.)
* "(The chicken) (is (ready to eat))." (The chicken itself is ready for consumption.)

"I saw the man on the hill with the telescope."

**Parse Trees:**

* "I saw ((the man) (on the hill)) (with the telescope)." (I used the telescope to see the man who was on the hill.)
* "I saw (the man) ((on the hill) (with the telescope))." (The man, who was on the hill, had the telescope.)

"Time flies like an arrow."

**Parse Trees:**

* "(Time flies) (like (an arrow))." (Time moves quickly in a manner similar to that of an arrow.)
* "((Time) (flies like) (an arrow))." (The concept of time is fond of arrows in general.)

"They are hunting dogs."

**Parse Trees:**

* "(They) (are (hunting dogs))." (They themselves are dogs that are hunting.)
* "(They are) (hunting dogs)." (They are pursuing dogs.)

In WordNet, a corpus is a collection of texts or documents that serve as a reference for creating synsets and calculating similarity between words. Here are some examples of corpus names and an overview of how synsets are created and how similarity between words is calculated in WordNet:

**Corpus Names:**

1. **WordNet:**
   * WordNet itself can be considered a corpus. It is a lexical database of English words organized into synsets (sets of synonyms) and linked by semantic relations.
2. **Brown Corpus:**
   * The Brown Corpus is a well-known text corpus of American English, which was the first million-word electronic corpus of English.
3. **Gutenberg Corpus:**
   * The Gutenberg Corpus is a collection of electronic texts of public domain books, primarily in English, made available by Project Gutenberg.
4. **Reuters Corpus:**
   * The Reuters Corpus is a collection of news articles published by Reuters, commonly used for text classification and information retrieval tasks.

**Synsets Creation in WordNet:**

* Synsets (sets of synonyms) in WordNet are created through the process of manual curation and lexical analysis by lexicographers.
* Each synset represents a distinct concept, and its members are words or phrases that are synonymous within that concept.
* Synsets are organized in a hierarchical structure, where more general concepts (hypernyms) are at the top and more specific concepts (hyponyms) are at the bottom.

**Similarity Calculation in WordNet:**

* WordNet provides measures of semantic similarity between words based on the structure of the synsets and their relationships.
* One common measure of similarity is based on the shortest path distance between synsets in the WordNet hierarchy. This is often computed using the shortest path length between two synsets in the hypernym/hyponym tree.
* Another measure of similarity is based on information content, which takes into account the frequency of usage of terms in a corpus. Terms that are less common are considered to have higher information content and may be more similar.
* Several algorithms, such as path similarity and Wu-Palmer similarity, are used to compute similarity scores between synsets based on these principles.

NLU (Natural Language Understanding) and NLG (Natural Language Generation) are two complementary aspects of Natural Language Processing (NLP) that deal with understanding and generating human language, respectively.

**Natural Language Understanding (NLU):**

NLU focuses on enabling computers to understand and interpret human language input in a way that captures the meaning and intent behind the text. Key tasks in NLU include:

1. **Speech Recognition:** Converting spoken language into text.
2. **Tokenization:** Breaking down text into smaller units such as words or phrases.
3. **Part-of-Speech Tagging (POS):** Identifying the grammatical parts of speech (e.g., noun, verb, adjective) of words in a sentence.
4. **Named Entity Recognition (NER):** Identifying and classifying named entities such as people, organizations, locations, etc., mentioned in text.
5. **Syntactic Parsing:** Analyzing the grammatical structure of sentences to understand the relationships between words.
6. **Semantic Analysis:** Understanding the meaning of words and sentences, including tasks such as sentiment analysis, word sense disambiguation, and semantic role labeling.
7. **Pragmatic Analysis:** Interpreting the context and intentions behind language use, including tasks such as discourse analysis and speech act recognition.

NLU enables systems to comprehend and process natural language input, allowing them to respond appropriately or take relevant actions based on the understood meaning.

**Natural Language Generation (NLG):**

NLG focuses on generating human-like language output from structured data or knowledge representations. Key tasks in NLG include:

1. **Text Planning:** Structuring and organizing the content to be generated.
2. **Content Selection:** Selecting the appropriate information or content to include in the generated text.
3. **Sentence Planning:** Planning the structure and composition of individual sentences.
4. **Lexicalization:** Mapping abstract concepts or meanings to specific words or phrases.
5. **Surface Realization:** Generating the final grammatically correct and fluent text output.

NLG systems can be used for various applications, including generating reports, summarizing data, creating natural language responses in chatbots, and generating personalized content.

1. **Tokenization:**
   * Tokenization is the process of breaking down a text into smaller units called tokens.
   * Tokens are typically words, phrases, or symbols that serve as the basic building blocks for further analysis.
   * For example, given the sentence "The quick brown fox jumps over the lazy dog," tokenization would result in individual tokens such as ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"].
2. **Stemming:**
   * Stemming is the process of reducing words to their base or root form, called the stem.
   * It involves removing suffixes from words to obtain the stem, which may not always be a valid word.
   * For example, stemming would convert words like "running," "runs," and "runner" into the base form "run."
3. **Lemmatization:**
   * Lemmatization is similar to stemming but aims to output valid words known as lemmas.
   * It involves reducing words to their dictionary form or lemma.
   * Lemmatization takes into account the morphological analysis of words and aims to return valid words with semantic meaning.
   * For example, lemmatization would convert words like "went" and "gone" into the single lemma "go," or "better" and "best" into the lemma "good."
4. **POS Tags (Part-of-Speech Tags):**
   * POS tagging is the process of assigning grammatical tags to each word in a sentence based on its syntactic role.
   * Common POS tags include noun (NN), verb (VB), adjective (JJ), adverb (RB), pronoun (PRP), preposition (IN), conjunction (CC), etc.
   * POS tagging helps in understanding the grammatical structure of sentences and identifying the roles of words in the sentence.
5. **Named Entity Recognition (NER):**
   * Named Entity Recognition is the process of identifying and classifying named entities (such as people, organizations, locations, dates, etc.) mentioned in text.
   * NER systems typically use machine learning algorithms or rule-based approaches to identify and classify named entities.
   * NER helps in extracting structured information from unstructured text and is useful in various applications such as information extraction, question answering, and named entity linking.

Word2Vec is a popular model used in natural language processing (NLP) to create vector representations of words, known as word embeddings. Developed by a team of researchers led by Tomas Mikolov at Google in 2013, Word2Vec has become a fundamental technique for understanding and processing human language computationally. The model aims to capture the semantic meaning of words based on their context within a corpus of text.

**Key Concepts of Word2Vec**

1. **Word Embeddings**: These are dense vector representations of words in a continuous vector space. Each word is represented by a unique vector of fixed size, typically between 100 and 300 dimensions. The goal is for words with similar meanings to have similar vectors.
2. **Context**: The context of a word is defined by its neighboring words in a sentence or a text. For example, in the sentence "The cat sat on the mat," the context for the word "cat" could be "The," "sat," "on," "the," and "mat."

**How Word2Vec Works**

Word2Vec uses two main model architectures:

1. **Continuous Bag of Words (CBOW)**: This architecture predicts the target word (the center word) based on its surrounding context words. For example, given the context words "The," "sat," "on," "the," and "mat," the CBOW model tries to predict the target word "cat."
2. **Skip-Gram**: This architecture works in the opposite way. It predicts the context words given the target word. For example, given the target word "cat," the Skip-Gram model tries to predict the context words "The," "sat," "on," "the," and "mat."

**Training Word2Vec**

During training, Word2Vec learns to adjust the vectors in such a way that words appearing in similar contexts have similar vectors. This is achieved through a neural network with a single hidden layer. The training process involves:

1. **Input Layer**: Represents the target word or context words.
2. **Hidden Layer**: Contains the word vectors that are adjusted during training.
3. **Output Layer**: Predicts the context words (in Skip-Gram) or the target word (in CBOW).

The objective is to maximize the probability of correctly predicting the context words given the target word (Skip-Gram) or vice versa (CBOW).

**Applications of Word2Vec**

* **Semantic Similarity**: Since words with similar meanings have similar vectors, Word2Vec can be used to measure semantic similarity between words.
* **Text Classification**: Word embeddings can serve as input features for text classification tasks, such as sentiment analysis.
* **Clustering and Visualization**: Word vectors can be clustered and visualized to reveal relationships between words and concepts.

**Advantages of Word2Vec**

* **Efficiency**: Word2Vec is computationally efficient and can be trained on large datasets relatively quickly.
* **Quality of Embeddings**: It produces high-quality embeddings that capture subtle semantic relationships between words.

**Limitations of Word2Vec**

* **Context Independence**: Word2Vec generates a single vector for each word, which means it doesn't account for polysemy (words with multiple meanings) based on different contexts.
* **Lack of Structure**: While Word2Vec captures semantic similarity, it doesn't capture syntactic structures well.

TF-IDF, which stands for Term Frequency-Inverse Document Frequency, is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (or corpus). It is widely used in information retrieval and text mining to identify significant words in documents and to aid in tasks such as keyword extraction and document similarity comparisons.

A Hidden Markov Model (HMM) is a statistical method that deals with understanding a system where we can only observe the **effects** (outputs) of the system, but not its internal **states**. HMMs are widely used in various applications like speech recognition, text analysis, bioinformatics, and more.

Here's a breakdown of the key concepts in HMMs:

* **Hidden States:** These are the underlying states of the system that we cannot directly observe. They represent different conditions or situations the system can be in. For example, in speech recognition, hidden states could represent whether the speaker is pronouncing a vowel or a consonant.
* **Emissions:** These are the observable outputs of the system. They depend on the hidden state the system is currently in. In speech recognition, emissions would be the actual sounds we hear.
* **Transition Probabilities:** These probabilities represent the likelihood of the system transitioning from one hidden state to another. For instance, the probability of moving from a vowel state to a consonant state in speech recognition.
* **Emission Probabilities:** These probabilities represent the likelihood of observing a particular emission given that the system is in a specific hidden state. For example, the probability of hearing a specific sound (emission) given that the speaker is currently in a vowel state (hidden state).

An HMM uses these probabilities to achieve two main goals:

1. **Decoding:** Given a sequence of emissions (observations), determine the most likely sequence of hidden states that generated those emissions. In speech recognition, this would be like figuring out the most likely sequence of phonemes (hidden states) that produced the speech audio (emissions).
2. **Learning:** Given a set of emission sequences and their corresponding hidden state sequences (training data), estimate the transition and emission probabilities of the HMM. This training allows the HMM to learn the characteristics of the system it's modeling.

The Viterbi algorithm is a dynamic programming technique used to find the most likely sequence of hidden states that explains a sequence of observed events in a Hidden Markov Model (HMM). It's particularly useful in decoding applications like speech recognition, text analysis, and bioinformatics.

Here's a breakdown of the Viterbi algorithm:

**Understanding HMMs:**

* HMMs deal with systems where we can only observe the outputs (emissions) and not the underlying states (hidden states) that generate those outputs.
* The Viterbi algorithm operates on an HMM with defined:
  + Hidden states: Represent the underlying conditions of the system (e.g., Sunny/Rainy weather).
  + Emissions: Represent the observable outputs (e.g., Sunshine/Rain).
  + Transition probabilities: Likelihood of transitioning between hidden states (e.g., Sunny to Rainy).
  + Emission probabilities: Likelihood of observing an emission given a hidden state (e.g., Sunshine on a Sunny day).

**What the Viterbi Algorithm Does:**

Given a sequence of observed emissions (e.g., Sun -> Rain -> Sun), the Viterbi algorithm finds the most likely sequence of hidden states (e.g., Sunny -> Rainy -> Sunny) that could have generated those emissions based on the HMM's transition and emission probabilities.

**Steps of the Viterbi Algorithm:**

1. **Initialization:**
   * For each hidden state at the first time step (t=1), calculate the probability of observing the first emission (e.g., Sun) given that the system is in that state.
   * Initialize a path probability variable (Viterbi path probability) for each hidden state at t=1, representing the highest probability of reaching that state with the observed emission sequence up to that point.
2. **Recursion:**
   * Iterate through each subsequent time step (t=2, 3, ...) in the emission sequence.
   * For each hidden state at the current time step:
     + Calculate the probability of transitioning to that state from each possible hidden state in the previous time step, considering the transition probabilities.
     + Multiply the transition probability by the Viterbi path probability of the previous state it came from.
     + Consider all possible previous states and choose the one with the highest resulting probability.
     + Update the Viterbi path probability for the current hidden state with this maximum value.
     + Keep track of the previous state (hidden state at t-1) that led to the highest probability for the current state (useful for backtracking later).
3. **Termination:**
   * After processing all emissions, the Viterbi algorithm reaches the final time step with Viterbi path probabilities for each hidden state.
4. **Backtracking:**
   * Identify the final hidden state with the highest Viterbi path probability.
   * Backtrack through the chosen previous states at each time step, following the track kept during recursion. This path represents the most likely sequence of hidden states.

CountVectorizer is a feature extraction technique provided by the scikit-learn library in Python. It converts a collection of text documents into a matrix of token counts, essentially representing the frequency of words in each document. This method is widely used in Natural Language Processing (NLP) to prepare textual data for machine learning algorithms.

How CountVectorizer Works

Tokenization: Splits the text into words (tokens).

Vocabulary Building: Identifies the unique words (vocabulary) across the entire dataset.

Vectorization: Creates a matrix where each row represents a document and each column represents a word from the vocabulary. The value in each cell is the count of the word in the corresponding document.

1. **High Dimensionality**:
   * **Sparse Matrix**: The resulting matrix from CountVectorizer is often sparse, meaning most entries are zero because many words do not appear in most documents. This can lead to inefficiencies in memory usage and computation.
   * **Curse of Dimensionality**: High-dimensional data can be difficult to manage and may degrade the performance of machine learning models, especially with limited computational resources.
2. **Ignores Semantic Meaning**:
   * **No Context**: CountVectorizer does not capture the semantic meaning of words. It treats each word as independent and ignores the context in which words appear. For instance, it cannot differentiate between homonyms (e.g., "bank" as a financial institution vs. "bank" of a river).
3. **No Handling of Synonyms**:
   * **Vocabulary Size**: It treats synonyms as different features, which can unnecessarily increase the size of the vocabulary and lead to redundancy in the feature set.
4. **Inability to Capture Relationships Between Words**:
   * **Word Order**: CountVectorizer disregards the order of words. As a result, it cannot capture phrases or n-grams effectively unless explicitly configured to do so.
5. **Stop Words and Common Words**:
   * **Frequent Words**: Common words (e.g., "the", "is") may dominate the feature space, adding noise and not providing useful information for distinguishing between documents.
   * **Stop Words**: While you can configure CountVectorizer to ignore stop words, this requires additional preprocessing steps.
6. **Sensitivity to Variations**:
   * **Case Sensitivity**: By default, CountVectorizer treats "Word" and "word" as different features, though this can be adjusted with configuration.
   * **Misspellings and Variations**: Misspelled words or different forms of the same word (e.g., "run" vs. "running") are treated as separate features, leading to data fragmentation.

**Hyponymy:**

* **Definition:** Hyponymy refers to a hierarchical relationship between words, where one word (hyponym) is a more specific type of another word (hypernym). It's a "kind-of" relationship.
* **Example:**
  + Hypernym: Furniture
  + Hyponyms: Chair, table, sofa

In this example, "furniture" is a general category, and "chair," "table," and "sofa" are all specific types of furniture.

**Homonymy:**

* **Definition:** Homonymy refers to words that have the same spelling and pronunciation but completely unrelated meanings. They are essentially different words that just happen to share the same form.
* **Example:**
  + Bat (flying mammal)
  + Bat (sporting equipment)

Here, "bat" refers to two entirely different things with distinct origins and meanings.

**Polysemy:**

* **Definition:** Polysemy refers to a single word having multiple related meanings. The word's core meaning branches out into different applications or contexts.
* **Example:**
  + Bank
  + Financial institution where you deposit money (e.g., I went to the bank to withdraw cash)
  + The edge of a river (e.g., We sat by the riverbank enjoying the view)

In this case, "bank" has two distinct meanings, but they are connected by a shared underlying concept of an edge or boundary (financial institution holding your money or the edge of the water).

Here's a table summarizing the key differences:

pen\_spark

Precision and recall are fundamental metrics used to evaluate the performance of search strategies, particularly in information retrieval (IR) systems. Here's a breakdown of how they work:

**Precision:**

* **Focuses on:** The **accuracy** of the retrieved results.
* **Calculation:**
  + Precision = (Number of relevant documents retrieved) / (Total number of documents retrieved)
* **Interpretation:** A high precision indicates that most of the retrieved documents are actually relevant to the user's query.

**Recall:**

* **Focuses on:** The **completeness** of the retrieved results.
* **Calculation:**
  + Recall = (Number of relevant documents retrieved) / (Total number of relevant documents in the collection)
* **Interpretation:** A high recall indicates that the search strategy has captured most of the relevant documents within the collection.

**Precision vs. Recall Trade-off:**

There's often a trade-off between precision and recall. Here's why:

* **Increasing precision** (focusing on highly relevant results) might lead to **lower recall** (missing some relevant documents).
* **Increasing recall** (aiming to capture all relevant documents) might lead to **lower precision** (retrieving some irrelevant documents).

The ideal balance between precision and recall depends on the specific search task and user needs.

* In some cases, users might prioritize high precision (e.g., legal research), where every retrieved document needs to be highly relevant.
* In other cases, maximizing recall might be more important (e.g., news search), where users want to see all potentially relevant articles, even if some turn out to be less relevant upon closer inspection.

Sentiment analysis, also known as opinion mining, can be categorized into two main types based on the level of detail it extracts from the text data:

1. **Polarity Classification:**
   * **Focus:** Overall emotional tone of the text data.
   * **Output:** Classifies the sentiment as positive, negative, or neutral.
   * **Applications:**
     + Gauging general customer satisfaction in reviews.
     + Identifying the overall sentiment of social media conversations about a brand or product.
2. **Fine-Grained Sentiment Analysis:**
   * **Focus:** More granular analysis of sentiment, including aspects, emotions, and intensity.
   * **Sub-categories:**
     + **Aspect-Based Sentiment Analysis (ABSA):**
       - Identifies the sentiment (positive, negative, neutral) and the target (aspect) of the opinion.
       - **Example:** "The food was delicious, but the service was slow." (ABSA can pinpoint "food" as positive and "service" as negative).
     + **Emotion Detection:**
       - Goes beyond basic sentiment to identify specific emotions expressed in the text, like anger, sadness, or joy.
     + **Intensity Detection:**
       - Analyzes the strength of the sentiment, classifying it as weak, moderate, or strong.