CBCS SCHEME

USN | | | | | | | | | | |

BAD613B

Sixth Semester B.E./B.Tech. Degree Examination, June/July 2025
Natural Language Processing

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
Q.1	a.	Define NLP. List and explain application of NLP.	10	L2	CO1
	b.	Describe Paninian Framework for Indian languages. Explain layered representation of Paninian Grammar and Karkce theory.	10	L3	CO1
		OR			
Q.2	a.	Explain transformational grammar with example.	10	L2	CO1
	b.	Consider the following corpus of three sentences. There is a big graden Children play in a graden They play inside beautiful garden Calculate P for the sentence "They play in a big Garden" assuming a Gram Language Model.	10	L3	CO1
		Module – 2			
Q.3	a.	Explain minimum edit distance algorithm. Compute minimum edit distance between "Tutor" and "tumor".	10	L3	CO2
	b.	List POS tagging methods. Explain rule based tagger with example.	10	L2	CO2
		OR			
Q.4	a.	What is morphological parsing? Explain 2-level morphological model with an example.	10	L2	CO2
	b.	Derive a top-down and bottom-up parse tree for the given sentence. "The angry bear chased the frightened little squirrel" Use the following grammar rule to create the parse tree. S → NP VP Det → the NP → Det Nom Adj →little/angry/frightened VP → V NP N → squirrel/bear Nom → Adj N V → Chased.	10	L3	CO2
		Module – 3			
Q.5	a.	Explain the working of the Naïve. Bayes classification algorithm with an example.	10	L2	CO3
	b.	Given the following short movie review, each labeled with a genre, either comedy or action. Fun, couple, love, love comedy Fast couple, shoot action Couple, fly, fast, fun, fun comedy Fly, fast, shoot, fun action Fly, fast, shoot, love action And a new document D' fast, couple, shoot, fly. Compute the most likely for D. Assume a Naïve bayes classifier and use add -1 smoothing for the likelihoods.	10	L3	CO3

		OR			
Q.6	a.	Explain multi nominal Naive Bayes classifiers briefly.	10	L2	CO3
	b.	Write short notes on : i) Training Navie Bayes classifiers ii) Optimizing for sentiment analysis.	10	L2	CO3
		Module – 4			
Q.7	a.	Explain design features of IR with a neat diagram.	10	L2	CO4
	b.	Explain the cluster and fuzzy model of IR system.	10	L2	CO4
		OR			
Q.8	a.	List different IR models. Explain classical information retrieval model.	10	L2	CO4
	b.	Explain WordNet with its applications.	10	L2	CO4
		Module – 5			
Q.9	a.	What is machine translation? Explain word order typology in detail with an example.	10	L2	CO5
	b.	What is mean by the automatic evaluation? Explain the automatic evaluation of character overlap chrF.	10	L2	CO5
		OR			
Q.10	a.	Describe the simple encoder-decoder model and identify the attention mechanism of the given sentence "The girl liked the pink frock".	10	L2	CO5
	b.	Write a brief note on bias and word embedding models.	10	L2	CO5

* * * * *

Q.1

a) Definition and Applications of NLP
Natural Language Processing (NLP) is a subfield of Artificial Intelligence (AI) focused on the interaction between computers and human (natural) languages. It enables machines to process and understand human language in a manner that is both valuable and meaningful. NLP is essential for applications that aim to bridge the gap between human communication and machines, allowing for the automation of processes that involve understanding, generation, and translation of language.

Key **applications of NLP** include:

- **Machine Translation**: Tools like Google Translate leverage NLP to automatically translate text from one language to another by using algorithms to match patterns in parallel corpora of translated text.
- **Speech Recognition**: NLP is used in applications like Apple's Siri, Google Assistant, and Amazon Alexa to transcribe spoken language into text. This is crucial for voice-based interaction.
- **Text-to-Speech (TTS)**: Converts written text into spoken words. This is widely used in applications like audiobooks, screen readers, and automated customer service systems.
- **Sentiment Analysis**: NLP helps businesses and social media platforms track opinions and emotions in customer reviews, tweets, and comments to gauge sentiment (positive, negative, or neutral).
- **Named Entity Recognition (NER)**: This involves identifying and classifying named entities in text into categories such as persons, organizations, locations, dates, etc. It is useful in extracting structured data from unstructured text.
- **Text Summarization**: NLP is used to automatically generate concise summaries of long articles, research papers, or news stories.
- **Chatbots and Virtual Assistants**: NLP is integral to chatbots (like those used in customer service), which understand user inputs in natural language and respond appropriately.

As NLP continues to evolve, its applications are becoming more sophisticated, with deeper language understanding and generation capabilities.

---

b) Paninian Framework for Indian Languages

The **Paninian framework** is based on the principles found in **Panini's Astadhyayi**, an ancient Sanskrit grammar text. It provides a formal system for analyzing the structure of Indian languages, particularly focusing on **syntax** and **morphology**. This framework has been adapted for modern computational linguistics to handle the complexities of Indian languages such as Hindi, Tamil, Bengali, etc.

In the Paninian model, language is represented at **three levels**:

1. **Surface Structure**: This is the actual form of the sentence, containing words and their grammatical markers such as tense, number, person, etc. For example, in the sentence “Ram plays,” the surface structure would include the words "Ram" and "plays" with their associated morphological markers.
2. **Intermediate Structure**: This stage abstracts away from the actual words and presents deeper syntactic relationships that are not immediately visible. It focuses on phrase structures and their relations.
3. **Deep Structure**: This is the most abstract level and represents the underlying meaning of the sentence. It’s more conceptual and independent of the surface form of the sentence.

The **Karkee Theory** is part of this framework and is concerned with syntactic transformations. In this theory, word forms are manipulated based on rules to ensure they fit the grammatical structure of a language. It emphasizes systematic derivation of word structures, especially in languages like Sanskrit, where the word

forms change according to their syntactic position in the sentence. This transformation is especially important for languages with complex inflection systems, such as those in the Indian subcontinent.

Q.2

a) Transformational Grammar

Transformational Grammar, proposed by Noam Chomsky, is a framework for understanding the structure of language by focusing on how sentences can be transformed or rearranged without changing their core meaning. It emphasizes the distinction between **surface structure** (the outward form of a sentence) and **deep structure** (the abstract, underlying meaning). Transformational rules describe how one sentence structure can be converted into another, often by moving words or elements around. For example, starting with a declarative sentence like "She is reading a book," we can apply a transformation to create the question "Is she reading a book?" by moving the auxiliary verb "is" to the beginning.

These transformations allow for different sentence forms such as questions, negations, and passive constructions while maintaining the same underlying meaning. For instance, from the sentence "She saw the man," one could apply a transformation to form the passive voice: "The man was seen by her." Transformational grammar is important for capturing linguistic variation and syntactic flexibility.

---

b) Corpus and Language Model Calculation

In the given corpus of three sentences:

1. "There is a big garden"
2. "Children play in a garden"
3. "They play inside a beautiful garden"

We are tasked with calculating the probability PPP for the sentence "They play in a big garden" using a **gram language model**.

A **gram language model** (such as an **N-gram model**) calculates the probability of a sentence by looking at the likelihood of a word appearing given its preceding words. In an N-gram model, we focus on the conditional probability of the next word in a sequence based on the previous N-1 words.

For example, in a bigram model, the probability of a sentence is calculated by multiplying the probabilities of each pair of consecutive words. For the sentence "They play in a big garden," we would calculate the probability for each word pair (e.g., "They play," "play in," "in a," etc.). These probabilities are estimated by counting how often each word pair appears in the given corpus and dividing by the frequency of the first word in the pair.

Thus, the probability PPP for "They play in a big garden" is the product of the probabilities of all word pairs in the sentence:

P("They play in a big garden")=P("They" | start)×P("play" | They)×P("in" | play)×P("a" | in)×P("big" | a)×P("garden" | big)P(\text{"They play in a big garden"}) = P(\text{"They" | start}) \times P(\text{"play" | They}) \times P(\text{"in" | play}) \times P(\text{"a" | in}) \times P(\text{"big" | a}) \times P(\text{"garden" | big})P("They play in a big garden")=P("They" | start)×P("play" | They)×P("in" | play)×P("a" | in)×P("big" | a)×P("garden" | big)

This approach enables the model to capture dependencies between words and their likelihood based on the observed frequencies within the corpus.

Q.3

a) Minimum Edit Distance Algorithm
The **minimum edit distance** algorithm, also known as **Levenshtein distance**, calculates the minimum number

of operations (insertions, deletions, and substitutions) needed to transform one string into another. The algorithm works by constructing a matrix that represents the cumulative cost of transforming one string into the other. The operations considered are:

- **Insertion**: Inserting a character at a specific position.
- **Deletion**: Removing a character.
- **Substitution**: Replacing one character with another.

The steps to compute the minimum edit distance between two words, such as "Tutor" and "tumor", are as follows:

1. Create a matrix where the size is (m+1)×(n+1)(m+1) \times (n+1)(m+1)×(n+1), where **m** and **n** are the lengths of the two strings.
2. Initialize the first row and column with incremental values (representing the cost of transforming between empty strings and the words).
3. Compare characters in each string. For each pair of characters (one from each string), if they match, no substitution is needed, and the value is copied from the diagonal. If they do not match, the minimum cost of insertion, deletion, or substitution is chosen.
4. The final value in the bottom-right corner of the matrix gives the minimum edit distance.

For "Tutor" and "tumor":

- Initial strings: "Tutor" = T, u, t, o, r and "tumor" = t, u, m, o, r
- After running the algorithm, the minimum edit distance is 2 (one substitution: "T" to "t" and one insertion: "m").

---

b) POS Tagging Methods & Rule-Based Tagger
POS (Part-of-Speech) tagging assigns a grammatical category (e.g., noun, verb, adjective) to each word in a sentence. Several methods are used to perform POS tagging:

- **Rule-Based Tagging**: Involves hand-crafted rules based on patterns in the input data. These rules often rely on local context, such as the preceding word or part of speech, to decide the tag.
- **Statistical Tagging**: This uses probabilistic models like Hidden Markov Models (HMM) to determine the most likely tag based on the context.
- **Machine Learning-Based Tagging**: Supervised learning algorithms, such as decision trees or neural networks, are trained on labeled data to predict the correct POS tag for each word.
- **Transformation-Based Tagging**: Also known as Brill tagging, it iteratively refines tags by applying transformation rules.

A **rule-based tagger** works by using predefined rules to tag words. For example, if the word is preceded by an article (e.g., "the"), the tagger might assign the POS tag "noun" to the word. Here's a simple rule:

- Rule: If a word follows "the", tag it as a noun (e.g., "the dog" → dog is tagged as a noun).
  This approach, though simple, is effective when dealing with well-defined contexts but may struggle with ambiguous cases.

Q.4

a) Morphological Parsing & 2-Level Morphological Model
Morphological parsing involves analyzing the structure of words to understand their components, such as

roots, prefixes, and suffixes. It is a crucial step in natural language processing for languages with rich morphology (e.g., inflection, derivation).

A **2-level morphological model** has two components:

1. **Lexical Level**: This involves identifying morphemes, the smallest units of meaning, such as roots, prefixes, and suffixes.
2. **Surface Level**: This transforms the lexical form into the actual surface form used in sentences, accounting for affixation, compounding, or other morphological changes.

Example:

- **Word**: "runners"
    - Lexical level: root = "run", affix = "-er", "-s"
    - Surface level: "runners" as used in the sentence.

---

b) Top-Down and Bottom-Up Parse Tree

Top-Down Parsing starts from the highest level of the parse tree (the root) and breaks down the sentence into sub-structures. It applies grammar rules recursively until it reaches the terminal symbols (words).

Bottom-Up Parsing starts with the terminal symbols and builds up to the root of the parse tree by combining smaller units into larger ones according to the grammar rules.

Given the sentence **"The angry bear chased the frightened little squirrel"** and the provided grammar rules, the parse trees are as follows:

- **Top-Down Parsing**:
    1. Start with S→NP VPS \to NP \ VPS→NP VP
    2. Break down NPNPNP into Det NomDet \ NomDet Nom
    3. Continue breaking down each component:
        - Det→theDet \to theDet→the
        - Nom→Adj NNom \to Adj \ NNom→Adj N
        - Adj→angryAdj \to angryAdj→angry, N→bearN \to bearN→bear
    4. Break down VPVPVP:
        - VP→V NPVP \to V \ NPVP→V NP
        - V→ChasedV \to ChasedV→Chased
        - Break down the second NPNPNP:
            - Det→theDet \to theDet→the
            - Nom→Adj NNom \to Adj \ NNom→Adj N
            - Adj→frightenedAdj \to frightenedAdj→frightened, N→squirrelN \to squirrelN→squirrel
- **Bottom-Up Parsing**:
    1. Start with words and combine them into phrases:
        - "the" → DetDetDet

- "angry" → AdjAdjAdj
- "bear" → NNN, combine to form NomNomNom

2. Combine components into larger structures:
    - Det NomDet \ NomDet Nom → NPNPNP
    - "chased" → VVV, combine with the second NPNPNP
3. Ultimately combine NPNPNP and VPVPVP into SSS.

Both methods lead to the same final parse tree, but the process differs in their approach to building the structure.

Q.5

a) Naïve Bayes Classification Algorithm

The **Naïve Bayes classifier** is a probabilistic classifier based on **Bayes' theorem**, which assumes that the features (predictors) are independent of each other given the class label. This "naïve" assumption of independence simplifies the computation of the conditional probability of each class, given the features.

The core of the Naïve Bayes classifier involves computing the posterior probability of each class given the input features (e.g., words in a document). Bayes' theorem is used to compute the probability as follows:

P(C|X)=P(X|C)·P(C)P(X)P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}P(C|X)=P(X)P(X|C)·P(C)

Where:

- P(C|X)P(C|X)P(C|X) is the **posterior probability** of the class CCC given the input features XXX.
- P(X|C)P(X|C)P(X|C) is the **likelihood** of observing the features XXX given the class CCC.
- P(C)P(C)P(C) is the **prior probability** of the class CCC.
- P(X)P(X)P(X) is the **evidence** or the probability of the features XXX, which acts as a normalizing constant.

To classify a new instance, the Naïve Bayes classifier computes the posterior probabilities for all classes and selects the class with the highest probability.

For example, in the context of spam email classification:

1. **Prior probability**: The probability that an email is spam, P(spam)P(\text{spam})P(spam), or not spam, P(ham)P(\text{ham})P(ham), based on the overall dataset.
2. **Likelihood**: The probability of seeing certain words (e.g., "offer," "discount") in spam vs. ham emails.
3. **Posterior probability**: Using Bayes' theorem, we compute which class (spam or ham) is most likely for a given email.

In the **text classification** example, Naïve Bayes assumes that each word's occurrence is independent of others, which makes it computationally efficient.

---

b) Movie Genre Classification Using Naïve Bayes

Let's apply the **Naïve Bayes classifier** to the movie reviews to determine the most likely genre for the new document D′D'D′: "fast, couple, shoot, fly."

We are given the following data:

- Reviews with labels (comedy or action):

- **Review 1**: "Fun, couple, love, love" → **comedy**
- **Review 2**: "Fast, couple, shoot" → **action**
- **Review 3**: "Couple, fly, fast, fun" → **comedy**
- **Review 4**: "Fly, fast, shoot, love" → **action**

We are tasked with predicting the genre of the new review D′D'D′: "fast, couple, shoot, fly."

We will use **add-1 smoothing** (also known as **Laplace smoothing**) to handle zero probabilities for unseen words. The formula for the Naïve Bayes classifier with smoothing is:

P(C|X)=P(C)·∏P(Xi|C)P(X)P(C|X) = \frac{P(C) \cdot \prod P(X_i | C)}{P(X)}P(C|X)=P(X)P(C)·∏P(Xi|C)

Where P(Xi|C)P(X_i | C)P(Xi|C) is the probability of each feature XiX_iXi (i.e., word) given the class CCC, smoothed with Laplace:

P(Xi|C)=count(Xi,C)+1count(C)+|V|P(X_i | C) = \frac{\text{count}(X_i, C) + 1}{\text{count}(C) + |V|}P(Xi|C)=count(C)+|V|count(Xi,C)+1

Where:

- count(Xi,C)\text{count}(X_i, C)count(Xi,C) is the number of times word XiX_iXi appears in class CCC.
- count(C)\text{count}(C)count(C) is the total number of words in class CCC.
- |V||V||V| is the size of the vocabulary (total distinct words).

Step 1: Calculate Prior Probabilities

- **Comedy**: There are 2 comedy reviews out of 4, so P(comedy)=24=0.5P(\text{comedy}) = \frac{2}{4} = 0.5P(comedy)=42=0.5.
- **Action**: There are 2 action reviews out of 4, so P(action)=24=0.5P(\text{action}) = \frac{2}{4} = 0.5P(action)=42=0.5.

Step 2: Calculate Likelihood for Each Class

We need to count the occurrences of each word in the classes "comedy" and "action." The vocabulary VVV consists of the words: **fun, couple, love, fast, shoot, fly**.

For Comedy:

- "Fun" appears 1 time.
- "Couple" appears 2 times.
- "Love" appears 2 times.
- "Fast" appears 1 time.
- "Shoot" appears 0 times.
- "Fly" appears 1 time.

For Action:

- "Fun" appears 0 times.
- "Couple" appears 1 time.
- "Love" appears 1 time.

- "Fast" appears 2 times.
- "Shoot" appears 2 times.
- "Fly" appears 1 time.

Step 3: Apply Add-1 Smoothing and Compute the Likelihoods for the New Document

For each class (comedy and action), calculate the probability for the document D′="fast, couple, shoot, fly"D' = \text{"fast, couple, shoot, fly"}D′="fast, couple, shoot, fly". Using the smoothed formula, for each word in D′D'D′, compute the likelihood in both classes. Multiply the priors and likelihoods for each class, and the class with the higher product is the predicted genre.

Q.6

a) Multinomial Naïve Bayes Classifier
The **Multinomial Naïve Bayes classifier** is a probabilistic machine learning algorithm widely used for text classification tasks. It is based on **Bayes' Theorem**, which calculates the probability of a class given the observed features. In the case of text classification, the features are typically words or tokens from the text, and the classes are the categories or labels that we aim to predict (e.g., spam or not spam in email classification).

The model assumes that the words (features) are conditionally independent given the class label, a simplifying assumption that makes the model "naïve." This assumption of independence, although not always true in practice, allows for efficient computation and works surprisingly well in many real-world applications.

The **Multinomial** aspect refers to the fact that we are working with the frequency of words rather than just their presence or absence. This means the model looks at the number of times each word appears in a document, and uses that count to calculate probabilities. Given a document, the algorithm computes the probability that the document belongs to each class based on the word frequencies in that document.

The core equation of the Naïve Bayes classifier is:

P(C|X)=P(C)⋅P(X|C)P(X)P(C|X) = \frac{P(C) \cdot P(X|C)}{P(X)}P(C|X)=P(X)P(C)⋅P(X|C)

Where:

- P(C|X)P(C|X)P(C|X) is the posterior probability of class CCC given features XXX,
- P(C)P(C)P(C) is the prior probability of class CCC,
- P(X|C)P(X|C)P(X|C) is the likelihood of features XXX given class CCC,
- P(X)P(X)P(X) is the evidence or the total probability of features XXX across all classes.

In a multinomial Naïve Bayes classifier, the likelihood P(X|C)P(X|C)P(X|C) is computed by assuming that each word in the document is generated independently from a multinomial distribution, and the probability of a word given a class is determined from the frequency of that word in documents of that class.

The process of training the multinomial Naïve Bayes model involves:

1. **Calculating prior probabilities** P(C)P(C)P(C) by dividing the number of documents in each class by the total number of documents.
2. **Calculating word likelihoods** P(wi|C)P(w_i | C)P(wi|C) by counting the occurrences of each word wiw_iwi in documents of class CCC and dividing by the total number of words in documents of that class.
3. **Classifying new documents** by calculating the posterior probability for each class and selecting the class with the highest probability.

---

b) Training Naïve Bayes Classifiers & Optimizing for Sentiment Analysis

i) Training Naïve Bayes Classifiers
Training a Naïve Bayes classifier for text classification involves the following steps:

1. **Prepare the training dataset**: Collect a labeled dataset where each text document has an associated class label (e.g., "positive" or "negative" for sentiment analysis).
2. **Preprocess the text**: Clean the text data by removing stopwords, punctuation, and stemming or lemmatizing the words to reduce them to their root forms.
3. **Calculate prior probabilities**: Estimate the prior probability for each class based on the proportion of documents belonging to each class.
4. **Calculate likelihoods**: Compute the likelihood of each word given a class. This is done by counting the occurrences of each word in the documents of a class and dividing by the total number of words in that class.
5. **Handle unseen words**: Apply smoothing techniques, such as **Laplace smoothing**, to handle words that are not present in the training set but might appear in the test set.
6. **Classify new documents**: Given a new document, calculate the posterior probability for each class using Bayes' Theorem and select the class with the highest probability.

ii) Optimizing for Sentiment Analysis
Optimizing a Naïve Bayes classifier for **sentiment analysis** involves fine-tuning the model to accurately predict the sentiment (positive, negative, or neutral) of text. The optimization process includes:

1. **Feature Engineering**: In sentiment analysis, the most common features are unigrams (single words) and bigrams (pairs of words). You can also consider higher-order n-grams or apply **TF-IDF (Term Frequency-Inverse Document Frequency)** to weigh words based on their importance.
2. **Text Preprocessing**: Removing noise such as special characters, converting all words to lowercase, removing stopwords, and handling negations (e.g., converting "not good" into "bad").
3. **Handling Imbalanced Data**: In some cases, there might be an imbalance between the number of positive and negative reviews. Techniques like **oversampling**, **undersampling**, or **adjusting class weights** can help address this imbalance.
4. **Cross-validation and Hyperparameter Tuning**: Use cross-validation to evaluate the performance of the model. Adjust parameters such as the smoothing parameter in the Naïve Bayes model to find the optimal setting.
5. **Model Evaluation**: Measure the model's performance using metrics like **accuracy**, **precision**, **recall**, and **F1-score**. For sentiment analysis, also consider **confusion matrices** to assess how well the model distinguishes between positive, negative, and neutral sentiments.

Optimizing the Naïve Bayes model for sentiment analysis can lead to better predictions, ensuring that the model correctly identifies the sentiment of unseen text, which is crucial for applications like product reviews, social media analysis, and customer feedback.

Q.7

a) Design Features of IR (Information Retrieval) Systems
Information Retrieval (IR) systems are designed to efficiently store, index, and retrieve relevant documents from large datasets in response to user queries. The key design features of an IR system are:

1. **Indexing**: An IR system needs an index to facilitate efficient searching. Indexing involves creating a data structure that maps words or terms to the documents in which they appear. A **term-document matrix** or an **inverted index** is commonly used.

2. **Query Processing**: The system processes user queries to match them against the indexed documents. This involves parsing the query, removing stopwords, and normalizing the terms (e.g., stemming or lemmatization).

3. **Ranking**: Once documents are retrieved, they need to be ranked based on their relevance to the query. The ranking can be determined using methods like **TF-IDF** (Term Frequency-Inverse Document Frequency) or cosine similarity.

4. **Relevance Feedback**: This feature allows the system to improve the results based on user feedback. If the user indicates that certain documents are relevant or irrelevant, the system can refine future search results.

5. **Performance and Efficiency**: An IR system must be able to handle large datasets and provide fast query responses. Efficient algorithms and indexing structures are necessary for scalability.

6. **User Interface**: A good IR system provides an intuitive interface for users to submit queries, review results, and provide feedback.

Q.7b) Cluster and Fuzzy Model of IR System

Cluster Model of IR:
The **Cluster Model** of Information Retrieval (IR) aims to organize documents into groups (or clusters) based on similarity. This method can improve the retrieval process by allowing the system to return clusters of related documents rather than individual documents. The basic idea behind clustering is that documents with similar content will be grouped together.

- **How Clustering Works in IR**: In the context of IR, documents are represented as vectors in a multi-dimensional space, where each dimension corresponds to a term (word). The similarity between documents is computed, often using **cosine similarity** or **Euclidean distance**, and documents that are close to each other in this space are grouped together into clusters. These clusters are created using clustering algorithms such as **K-means** or **Hierarchical clustering**. Once the clusters are formed, when a query is issued, the system can retrieve entire clusters rather than individual documents, improving retrieval efficiency.

- **Benefits of the Cluster Model**:
    - **Efficiency**: Reduces the number of documents that need to be retrieved for a query by returning entire clusters.
    - **Improved Search Results**: By grouping related documents, users are more likely to receive relevant results in response to their queries.
    - **Relevance**: Cluster models are particularly useful when users are interested in discovering a range of related documents rather than a single, isolated document.

- **Challenges**:
    - **Cluster Quality**: The quality of clusters is heavily dependent on the algorithm used for clustering and the feature selection process.
    - **Dynamic Data**: Handling dynamic, ever-changing datasets can be challenging, as clusters may need to be continually updated.

Fuzzy Model of IR:
The **Fuzzy Model** of IR utilizes fuzzy logic to assign degrees of relevance to documents rather than classifying them as either relevant or irrelevant. Traditional IR models, such as the Boolean and vector space models, typically offer binary outcomes (relevant or not), which might oversimplify the relevance judgment. Fuzzy models, on the other hand, allow for a more flexible approach.

- **How Fuzzy Logic is Applied in IR**: Fuzzy logic is used to represent the uncertainty or vagueness in document relevance. In a fuzzy IR system, documents are ranked based on a **degree of relevance**,

which is a value between 0 and 1. The relevance of a document to a query is not binary, but rather is evaluated as a fuzzy set, where a document can be partially relevant. For example, a document might be 0.7 relevant to a query, meaning it is somewhat relevant but not the most relevant.

- **Benefits of the Fuzzy Model**:
    - **Flexibility**: Fuzzy IR allows for nuanced judgments of relevance, unlike traditional models that rely on hard thresholds.
    - **Improved Results**: It can improve user satisfaction by returning documents that are partially relevant, which might have been excluded in traditional IR models.
- **Challenges**:
    - **Complexity**: Implementing fuzzy models requires a deeper understanding of fuzzy logic and can introduce computational overhead.
    - **Interpretability**: Users may find it harder to interpret the relevance scores, especially when dealing with subjective criteria.

Q.8a) List Different IR Models. Explain Classical Information Retrieval Model.

Different IR Models:
There are several models of Information Retrieval (IR), each offering a different approach to retrieving and ranking documents based on a query. The main IR models are:

1. **Boolean Model**: This is the simplest IR model, which uses Boolean operators (AND, OR, NOT) to match documents with the query. A document is either relevant or irrelevant based on exact term matching, with no degrees of relevance.
2. **Vector Space Model**: In this model, both documents and queries are represented as vectors in a multi-dimensional space. The model uses a mathematical measure (typically cosine similarity) to assess the similarity between the document vector and the query vector. This model allows for ranking documents based on their relevance to the query.
3. **Probabilistic Model**: This model assumes that there is an underlying probability distribution for document relevance. The system retrieves documents based on the probability that they are relevant to the query. One popular approach within this model is the **BM25** algorithm, which uses term frequency and document length to compute the relevance score.
4. **Latent Semantic Analysis (LSA)**: LSA is a mathematical approach that reduces the dimensionality of the document-term matrix using **Singular Value Decomposition (SVD)**. This helps to uncover hidden semantic structures in the data, improving retrieval performance by addressing issues such as synonymy and polysemy.
5. **Fuzzy Logic Models**: As discussed earlier, fuzzy logic models allow documents to be partially relevant, with relevance measured on a continuous scale between 0 and 1, rather than as binary relevant/irrelevant.
6. **Neural Network Models**: These models, including deep learning approaches, learn to retrieve relevant documents by training on large datasets, often using techniques like **Word Embeddings** or **Convolutional Neural Networks (CNNs)** to analyze text.

Classical Information Retrieval Model:
The **Classical IR model** refers to models that represent documents and queries as collections of terms (words) and retrieve documents based on their matching terms. The **Vector Space Model** is the most commonly used classical model. In this model, documents are represented as vectors in a multi-dimensional space, where each

dimension corresponds to a term from a predefined vocabulary. The proximity between a query and a document is calculated using a similarity measure, such as **cosine similarity**.

- **Term Frequency (TF)**: Measures how frequently a term appears in a document.
- **Inverse Document Frequency (IDF)**: Measures how important a term is by considering the number of documents containing the term.

The **TF-IDF** score is used to weigh the importance of terms within a document and is fundamental in determining the relevance of a document to a given query.

Q.8b) Explain WordNet with its Applications

WordNet is a lexical database of the English language that groups words into **synsets** (sets of synonyms) and links them based on semantic relationships such as **hyponymy** (is-a relationships), **hypernymy** (has-a relationships), and **meronymy** (part-whole relationships). WordNet was developed at Princeton University and is widely used in **Natural Language Processing (NLP)** and **Information Retrieval (IR)** systems.

- **Structure of WordNet**:
    - WordNet organizes words into groups of synonyms (synsets) and defines relationships between them.
    - Synsets are linked to each other through semantic relationships, allowing for navigation through the different meanings of a word. For example, "dog" and "canine" are synonyms in one synset, while "dog" and "animal" are connected through a hypernym relationship.
- **Applications of WordNet**:

1. **Word Sense Disambiguation**: WordNet helps in disambiguating words with multiple meanings. For example, the word "bank" can refer to a financial institution or the side of a river. WordNet helps determine the correct meaning based on context.

2. **Information Retrieval**: WordNet can improve query expansion by retrieving documents containing semantically related words. For example, a query for "car" can also retrieve documents containing "automobile" or "vehicle."

3. **Machine Translation**: In machine translation, WordNet assists in finding the appropriate translations for words by leveraging the semantic relationships between words in different languages.

4. **Text Classification**: WordNet is used in text classification tasks to improve feature extraction by considering the semantic similarity between terms.

5. **Sentiment Analysis**: WordNet can be used to identify sentiment by examining the meanings and relationships between words and their synonyms.

WordNet's ability to link words and their meanings makes it an indispensable tool in many NLP and IR applications, aiding in the development of more semantically aware systems.

Q.9a) What is Machine Translation? Explain Word Order Typology in Detail with an Example

Machine Translation (MT) refers to the use of computer software to automatically translate text from one language to another. MT has become a vital tool in breaking down language barriers and enabling communication between people who speak different languages. There are several types of machine translation, including:

1. **Rule-Based Machine Translation (RBMT)**: This approach relies on linguistic rules to perform translation. It uses a detailed set of grammar rules for both the source and target languages.

2. **Statistical Machine Translation (SMT)**: SMT models are based on probabilities derived from large corpora of text. The system learns which translations are most likely based on the statistical patterns of the words in the source and target languages.

3. **Neural Machine Translation (NMT)**: NMT uses deep learning techniques, particularly recurrent neural networks (RNNs) and transformers, to improve translation quality by learning directly from vast amounts of bilingual text data. It tends to generate more fluent and natural-sounding translations compared to rule-based and statistical models.

Word Order Typology:
Word order typology refers to the order in which the subject (S), verb (V), and object (O) typically appear in sentences in different languages. The word order plays a significant role in how machine translation systems handle linguistic structures. There are several common word order types, with the three main ones being:

1. **SVO (Subject-Verb-Object)**: This is the most common word order in English and many other languages. For example:
    - **English**: "The girl (S) likes (V) apples (O)."
2. **SOV (Subject-Object-Verb)**: This word order is prevalent in languages like Japanese, Korean, and Turkish. For example:
    - **Japanese**: "The girl (S) apples (O) likes (V)."
3. **VSO (Verb-Subject-Object)**: Used in languages such as Classical Arabic and Welsh. For example:
    - **Arabic**: "Likes (V) the girl (S) apples (O)."

Languages with different word orders can present challenges for machine translation. For example, translating a sentence from English (SVO) to Japanese (SOV) may require reordering the components of the sentence. NMT models handle such differences better by learning sentence patterns during training, allowing them to generate more contextually appropriate translations.

Word order typology is crucial because it helps machine translation systems understand the syntactic structure of both the source and target languages and make more accurate predictions when translating sentences.

Q.9b) What is Meant by Automatic Evaluation? Explain the Automatic Evaluation of Character Overlap (chrF)

Automatic Evaluation refers to the use of algorithms and computational methods to assess the quality of machine-generated translations without human involvement. In Machine Translation (MT), automatic evaluation metrics are crucial because manually evaluating translations is time-consuming and expensive. Various metrics are used to evaluate MT systems, with the most common ones being **BLEU (Bilingual Evaluation Understudy)**, **METEOR**, **ROUGE**, and **chrF**.

chrF is a metric specifically designed for evaluating the quality of character-based machine translations. Unlike traditional metrics such as BLEU, which operate on word-level n-grams, **chrF** operates at the **character level**. It measures the overlap of character n-grams between the machine-generated translation and a reference translation.

- **Why chrF?**: Many languages, especially morphologically rich ones, can present challenges for word-based metrics because the words can change forms through inflections or derivations. For instance, the translation of a word into a different language might result in small variations, such as added or removed suffixes, which chrF can capture at the character level, improving its evaluation capability. Additionally, chrF is particularly effective for languages with rich morphology (such as German, Russian, and Arabic), where word forms change frequently.

Character Overlap (chrF) measures how similar the machine translation is to a reference translation by computing the overlap of character n-grams (typically unigrams, bigrams, or trigrams) between the two translations. The formula for chrF is:

chrF=∑precision of character n-grams×recall of character n-gramslength of reference translationchrF = \frac{\sum \text{precision of character n-grams} \times \text{recall of character n-grams}}{\text{length of reference translation}}chrF=length of reference translation∑precision of character n-grams×recall of character n-grams

- **Precision**: This refers to the proportion of the character n-grams in the machine-generated translation that are also found in the reference translation.
- **Recall**: This refers to the proportion of the character n-grams in the reference translation that are also found in the machine-generated translation.

A higher **chrF score** indicates a closer match between the machine translation and the reference translation, meaning the translation is of higher quality.

Advantages of chrF:

1. **Character-Level Focus**: It works well for languages where words are often inflected or conjugated differently, helping to capture variations at the character level.
2. **Sensitivity to Word-Form Differences**: chrF is more sensitive to morphological variations, making it more reliable for evaluating translations in languages with complex morphology.
3. **Better Correlation with Human Evaluation**: Research has shown that chrF is often more closely correlated with human judgment than word-based metrics like BLEU, particularly for languages with rich morphology.

Q.10a) Describe the Simple Encoder-Decoder Model and Identify the Attention Mechanism of the Given Sentence

The **Encoder-Decoder model** is commonly used in **sequence-to-sequence tasks**, such as machine translation, speech recognition, and summarization. It consists of two main parts:

1. **Encoder**: The encoder takes the input sequence (e.g., a sentence in a source language) and processes it into a fixed-length context vector (a compact representation). The encoder typically uses a **Recurrent Neural Network (RNN)** or **Long Short-Term Memory (LSTM)** network to handle the sequential nature of the input data.
2. **Decoder**: The decoder takes the context vector from the encoder and generates the output sequence (e.g., a translated sentence). Like the encoder, the decoder is usually an RNN or LSTM. The decoder generates one token at a time (word or character) and uses the previously generated tokens and the context vector to predict the next token in the sequence.

Attention Mechanism: The traditional Encoder-Decoder model had limitations, especially when processing long sentences, because the encoder's fixed-length context vector could not capture the full context of long sequences. The **Attention Mechanism** overcomes this by allowing the model to "attend" to different parts of the input sequence while generating each token in the output sequence.

In **Attention**, for each token in the output sequence, the model assigns different attention weights to all the tokens in the input sequence. This enables the model to focus on the most relevant parts of the input sequence when generating each word in the output.

For the sentence **“The girl liked the pink frock”**, the attention mechanism would allow the model to focus on specific words during the translation process. For example, when generating the word “liked,” the attention mechanism might give more weight to the word "girl," as it is more directly related to the verb "liked."

Q.10b) Write a Brief Note on Bias and Word Embedding Models

Bias in Machine Learning:
Bias refers to systematic errors in a model that result in incorrect predictions or decisions, often due to unfair or skewed data. In machine learning, bias can arise from various sources:

- **Data Bias**: When the training data is unrepresentative of the real-world data the model will encounter, it can cause the model to favor certain outcomes. For instance, if a dataset used to train a sentiment analysis model contains mostly positive reviews, the model might be biased toward predicting positive sentiment.
- **Algorithmic Bias**: Sometimes, even if the data is fair, the algorithm itself might introduce bias. For example, some algorithms might overfit to certain patterns, leading to inaccurate generalizations.
- **Human Bias**: If the data is labeled or interpreted by humans, human biases can seep into the data, influencing how the model is trained and evaluated.

Word Embedding Models:
Word Embeddings are a type of **distributed representation** of words where words with similar meanings are represented by vectors that are close in the vector space. These embeddings are trained using deep learning techniques on large text corpora and can capture semantic relationships between words. Some popular word embedding models include:

1. **Word2Vec**: Uses a neural network to learn word representations by predicting neighboring words (Continuous Bag of Words or CBOW) or the target word given surrounding words (Skip-Gram).
2. **GloVe**: A matrix factorization-based model that captures global statistical information from a word co-occurrence matrix to generate word vectors.
3. **FastText**: An extension of Word2Vec that represents words as bags of character n-grams, which helps capture subword information and improves handling of out-of-vocabulary words.

While powerful, word embeddings can inherit **biases** from the data they are trained on, such as gender, racial, or ethnic biases, and these biases can affect downstream tasks like sentiment analysis, machine translation, or information retrieval.