USN | | | | | | | | | |

CMRIT

Internal Assessment Test 2 – JULY 2024

Sub:	Natural Language Processing					Sub Code:	21AI643	Branch:	AIML
Date:	08/07/2024	Duration:	90 mins	Max Marks:	50	Sem / Sec:	VI / A		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1	What are the different steps used to achieve the goal of case annotation? Case annotation is a crucial step in various fields such as law, medicine, and machine learning, where it involves the detailed labeling and documentation of cases for analysis and study. **Define Objectives and Scope**: • Determine the purpose of the annotation. • Identify the specific cases to be annotated. • Establish the criteria and guidelines for the annotation process. **Gather and Prepare Data**: • Collect relevant cases from various sources. • Ensure data quality and consistency. • Format the data appropriately for annotation (e.g., text, images, audio). **Develop Annotation Guidelines**: • Create detailed instructions for annotators to ensure consistency. • Define categories, tags, or labels to be used. • Provide examples and counterexamples. **Select and Train Annotators**: • Choose individuals with the necessary expertise. • Train them on the annotation guidelines. • Conduct pilot annotations to refine guidelines and ensure understanding. **Annotation Process**: • Assign cases to annotators. • Annotators label and document the cases according to guidelines. • Use annotation tools and software to facilitate the process. **Quality Control and Review**: • Implement inter-annotator agreement measures to check consistency. • Conduct spot checks and audits of annotations. • Provide feedback to annotators and make necessary corrections. **Data Integration and Analysis**: • Compile and integrate annotated cases into a central database. • Analyze the annotated data to extract insights and patterns.	[10] Description-6 Examples-4	CO3	L2

- Use the annotated data for the intended purpose (e.g., training machine learning models, legal case studies).

Documentation and Reporting:

- Document the entire annotation process.
- Report on the findings and outcomes of the annotation.
- Share insights with relevant stakeholders.

Continuous Improvement:

- Review and update guidelines based on feedback.
- Continuously monitor and improve the annotation process.
- Incorporate new cases and refine annotations as needed.

Examples:

Medical Field

Goal: To annotate medical images for a machine learning project to detect tumors.

1. **Define Objectives and Scope**:
   - Objective: Develop a model to identify tumors in MRI scans.
   - Scope: Annotate 1,000 MRI images.
2. **Gather and Prepare Data**:
   - Collect MRI images from hospitals.
   - Ensure images are in a compatible format (e.g., DICOM).
3. **Develop Annotation Guidelines**:
   - Define categories like "tumor," "benign mass," and "normal tissue."
   - Provide annotated examples of each category.
4. **Select and Train Annotators**:
   - Choose radiologists and medical imaging specialists.
   - Train them on the annotation tool and guidelines.
5. **Annotation Process**:
   - Annotators label regions of interest in each MRI scan.
   - Use specialized medical image annotation software.
6. **Quality Control and Review**:
   - Check inter-annotator reliability.
   - Review a subset of annotations for accuracy.
7. **Data Integration and Analysis**:
   - Integrate annotations into a dataset.
   - Use the dataset to train and validate the machine learning model.
8. **Documentation and Reporting**:
   - Document the annotation process and guidelines.
   - Report on the model's performance and findings.
9. **Continuous Improvement**:
   - Refine guidelines based on new medical knowledge.
   - Continuously improve the model with new data and annotations.

2	Write a short note on	[10]	CO3	L2

a. The shortest path hypothesis
b. Learning with dependency path.

Desc & examples
a-5
b-5

The Shortest Path Hypothesis

The shortest path hypothesis is a concept often used in natural language processing (NLP) and computational linguistics. It posits that the shortest syntactic or dependency path between two entities in a sentence often contains the most relevant information about their relationship. This hypothesis is based on the observation that the fewer intermediate nodes or words there are between two entities, the more direct and meaningful their connection is likely to be. By focusing on the shortest path, algorithms can efficiently extract significant relationships and reduce the noise that might come from longer, less direct paths.

For example, in a sentence like "The cat, which was sitting on the mat, chased the mouse," the shortest path between "cat" and "mouse" is directly through the verb "chased," bypassing the relative clause "which was sitting on the mat." This direct path captures the primary action relationship between the entities.

Learning with Dependency Path

Learning with dependency paths is an approach in NLP where models are trained to understand and utilize the syntactic structure of sentences to enhance their comprehension and task performance. Dependency paths represent the grammatical relationships between words in a sentence, forming a tree-like structure where nodes are words and edges represent syntactic dependencies.

By incorporating these paths, machine learning models can:

1. **Enhance Context Understanding**: Dependency paths help models to capture the context and relationships between words more effectively, improving tasks like relation extraction, sentiment analysis, and named entity recognition.
2. **Improve Feature Engineering**: Dependency paths provide rich, structured features that can be used in training machine learning algorithms, making them more robust and accurate.
3. **Reduce Complexity**: Focusing on relevant dependency paths allows models to ignore irrelevant parts of the sentence, reducing complexity and improving efficiency.

For example, in relation extraction, understanding the dependency path between two entities can help a model accurately determine how they are related. In the sentence "The CEO of the company announced a new policy," the dependency path from "CEO" to "policy" via "announced" helps in identifying the action taken by the CEO regarding the policy.

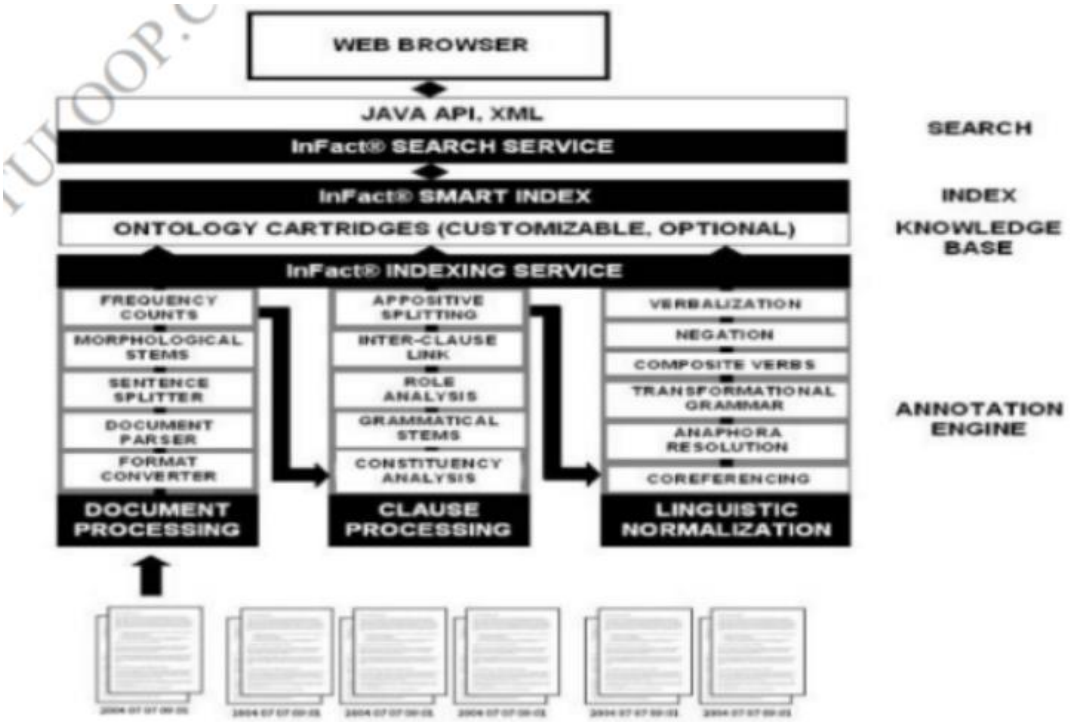
Steps to take features for supervised learning in relations between two named entities is

- The sequence of words between the new entities
- The part of speech tags of these words.
- Bag of words between the two words.

Sentence 1: “Rajesh is the owner of the company called, Seeland”
Sentence 2: "Reeta owns the black building called Heena”
From these two sentences, the vocabulary is as follows, assuming stop-words (ie, the, is, of, etc.) are

<table>
<tr>
<td></td>
<td>removed, and that you split the sentence into individual tokens:
vocabulary = {owner, company, called, owns, black, building}
By using the raw frequency of each word or any standard approach find the vectors related to a
sentence:
Sentence 1: {1, 1, 1, 0, 0, 0}
Sentence 2: {0, 0, 1, 1, 1, 1}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Write a python code on Functional overview of In-Fact System using COVID-19 Tracking System

Infact consists of an indexing and a search module. With reference to the given figure indexing pertains to the processing flow on the bottom of the diagram. InFact models text as a complex multivariate object using a unique combination of deep parsing, linguistic normalization, and efficient storage. The storage schema addresses the fundamental difficulty of reducing information contained in parse trees into generalized data structures that can be queried dynamically. In addition, InFact handles the problem of linguistic variation by mapping complex linguistic structures into semantic and syntactic equivalents. This representation supports dynamic relationship and event search, information extraction and pattern matching from large document collections in real time.



Document Processing:
・ The first step in document processing is format conversion, which is handled through native</td>
<td>[10]
Description-4
diagram-4
psuedocode-2</td>
<td>CO3</td>
<td>L3</td>
</tr>
</table>

format converters which can convert 370 different input file types.

・ Customized document parsers address the issue that a Webpage may not be the basic unit
of content, but it may consist of separate sections with an associated set of relationships and
metadata.
o For instance a blog post may contain blocks of text with different dates and topics.
o The challenge is to automatically recognize variations from a common style
template, and segment information in the index to match zones in the source
documents, so the relevant section can be displayed in response to a query.

・ Next we apply logic for sentence splitting in preparation for clause processing.

・ Last, we extract morphological stems and compute frequency counts, which are then entered in the index.

Clause Processing:

• The indexing service takes the output of the sentence splitter and feeds it to a deep linguistic
parser.

・ Indices are created automatically, without using predefined extraction rules, and it captures all
information, not just predefined patterns. The parser performs a full constituency and

dependency analysis, extracting part-of-speech (POS) tags and grammatical roles for all tokens in
every clause.

・ Next it captures inter-clause links, through:
o Explicit tagging of conjunction or pronouns that provide the link between the syntactic
structures for two adjacent clauses in the same sentence.
o Pointing to the list of annotated keywords in the antecedent and following sentence.

Linguistic Normalization:

・ Apply normalization rules at the syntactic, semantic, or even pragmatic level.

・ Our approach to coreferencing and anaphora resolution make use of syntactic agreement
and/or binding theory constraints.

・ Binding theory places syntactic restrictions on the possible coreference relationships between pronouns and their antecedents.
o Example:

・ "John works by himself," "himself" must refer to John.

・ "John bought him a new car," "him" must refer to some other individual mentioned in a previous sentence.

・ ""You have not been sending money," John said in a recent call to his wife
from Germany, "binding theory constraints limit pronoun resolution to first
and second. persons within a quotation (e.g., you)

・ Referencing and anaphora resolution models also benefit from preferential weighting based
on dependency attributes.

・ Apply a transformational grammar to map multiple surface structures into an equivalent deep structure.

・ A common example is the normalization of a dependency structure involving a passive verb

	form into the active, and recognition of the deep subject of such clause. ・ At the more pragmatic level, apply rules to normalize composite verb expressions, capture explicit and implicit negations, or to verbalize noun or adjectives ・ For instance, the sentences "Bill did not visit Jane," which contains an explicit negation, and "Bill failed to visit Jane, where the negation is rendered by a composite verb expression, are mapped to the same structure.			

```
import datetime

class COVID19TrackingSystem:
    def __init__(self):
        self.cases = []
        self.vaccination_records = []
        self.reports = []

    def add_case(self, case_id, date_reported, location, status):
        case = {
            'case_id': case_id,
            'date_reported': date_reported,
            'location': location,
            'status': status
        }
        self.cases.append(case)

    def add_vaccination_record(self, record_id, person_id, date_vaccinated, vaccine_type):
        record = {
            'record_id': record_id,
            'person_id': person_id,
            'date_vaccinated': date_vaccinated,
            'vaccine_type': vaccine_type
        }
        self.vaccination_records.append(record)

    def generate_report(self, report_date):
        total_cases = len(self.cases)
        total_vaccinated = len(self.vaccination_records)
        active_cases = len([case for case in self.cases if case['status'] == 'active'])
        recovered_cases = len([case for case in self.cases if case['status'] == 'recovered'])
        deaths = len([case for case in self.cases if case['status'] == 'deceased'])

        report = {
            'report_date': report_date,
            'total_cases': total_cases,
            'total_vaccinated': total_vaccinated,
            'active_cases': active_cases,
            'recovered_cases': recovered_cases,
            'deaths': deaths
        }
        self.reports.append(report)
```

```
        return report

    def display_report(self, report):
        print("COVID-19 Report as of", report['report_date'])
        print("Total Cases:", report['total_cases'])
        print("Total Vaccinated:", report['total_vaccinated'])
        print("Active Cases:", report['active_cases'])
        print("Recovered Cases:", report['recovered_cases'])
        print("Deaths:", report['deaths'])

# Example Usage
if __name__ == "__main__":
    system = COVID19TrackingSystem()

    # Adding COVID-19 cases
    system.add_case(1, datetime.date(2023, 7, 1), 'New York', 'active')
    system.add_case(2, datetime.date(2023, 7, 2), 'California', 'recovered')
    system.add_case(3, datetime.date(2023, 7, 3), 'Texas', 'deceased')
    system.add_case(4, datetime.date(2023, 7, 4), 'Florida', 'active')

    # Adding Vaccination Records
    system.add_vaccination_record(1, 'person_1', datetime.date(2023, 6, 1), 'Pfizer')
    system.add_vaccination_record(2, 'person_2', datetime.date(2023, 6, 2),
'Moderna')
    system.add_vaccination_record(3, 'person_3', datetime.date(2023, 6, 3), 'Johnson
& Johnson')

    # Generating and displaying report
    report = system.generate_report(datetime.date(2023, 7, 5))
    system.display_report(report)
```

- **COVID19TrackingSystem Class**: This class handles the tracking of COVID-19 cases and vaccination records. It also generates reports based on the data.
- **add_case Method**: Adds a new COVID-19 case to the system with details such as case ID, date reported, location, and status (e.g., active, recovered, deceased).
- **add_vaccination_record Method**: Adds a new vaccination record with details such as record ID, person ID, date vaccinated, and vaccine type.
- **generate_report Method**: Generates a report that summarizes the total number of cases, total vaccinated, active cases, recovered cases, and deaths as of a given date.
- **display_report Method**: Displays the generated report in a readable format.

4	a. Write an algorithm for the functioning of word matching feedback system used in ISTART. b. Write a note on various approaches to analyzing texts with examples.	[10] a)Desc- & examples-3 Algorithm-2	CO4	L3

	• iSTART (Interactive Strategy Training for Active Reading and Thinking) is an intelligent tutoring system designed to help students improve their reading comprehension skills. A word matching feedback system in iSTART can provide immediate feedback to users based on their input. Here's an algorithm for such a system:	b) Desc & examples-3 psuedocode -2		

Algorithm: Word Matching Feedback System

1. **Input**:
   - User input text (user_text)
   - Reference text (reference_text)
2. **Preprocessing**:
   - Tokenize `user_text` and `reference_text` into words.
   - Convert all tokens to lowercase to ensure case insensitivity.
   - Remove any punctuation or special characters from the tokens.
3. **Word Matching**:
   - Initialize an empty list `feedback` to store feedback messages.
   - For each word in the `reference_text` tokens:
     - Check if the word exists in the `user_text` tokens.
     - If the word exists, append a positive feedback message to `feedback`.
     - If the word does not exist, append a corrective feedback message to `feedback`.
4. **Additional Feedback (Optional)**:
   - Calculate the similarity score between `user_text` and `reference_text` using a similarity metric (e.g., cosine similarity, Jaccard similarity).
   - Provide additional feedback based on the similarity score (e.g., overall similarity percentage, areas for improvement).
5. **Output**:
   - Return or display the `feedback` list to the user.

Psuedocode

```
def word_matching_feedback_system(user_text, reference_text):
    # Preprocessing
    user_tokens = preprocess_text(user_text)
    reference_tokens = preprocess_text(reference_text)

    # Word Matching
    feedback = []
    for word in reference_tokens:
        if word in user_tokens:
            feedback.append(f"Good job! You used the word '{word}'.")
        else:
            feedback.append(f"Try to include the word '{word}'.")

    # Additional Feedback (Optional)
    similarity_score = calculate_similarity(user_text, reference_text)
    feedback.append(f"Overall similarity score: {similarity_score:.2f}")

    return feedback

def preprocess_text(text):
    # Tokenize, convert to lowercase, and remove punctuation
```

```
    tokens = text.lower().split()
    tokens = [token.strip('.,!?') for token in tokens]
    return tokens

def calculate_similarity(text1, text2):
    # Calculate similarity score between two texts (e.g., using cosine similarity)
    # This is a placeholder implementation
    return 0.85  # Example similarity score

# Example Usage
user_text = "The cat chased the mouse."
reference_text = "The cat was chasing a mouse."
feedback = word_matching_feedback_system(user_text, reference_text)
for message in feedback:
    print(message)
```

b)Approaches to Analyzing Texts with Examples

- **Statistical Methods**:
  - **Word Frequency Analysis**: Counting the occurrence of each word in a text. Commonly used to identify key themes or topics.
    - **Example**: Analyzing the frequency of words in a political speech to identify the main issues discussed.
  - **N-gram Analysis**: Examining contiguous sequences of n items (words) in a text. Useful for understanding common phrases or collocations.
    - **Example**: Using bigrams (2-grams) to study common two-word phrases in customer reviews.
- **Rule-Based Methods**:
  - **Regular Expressions**: Using pattern matching to identify specific text patterns. Useful for tasks like extracting dates, email addresses, or specific keywords.
    - **Example**: Extracting phone numbers from a document using regular expressions.
  - **Part-of-Speech (POS) Tagging**: Assigning POS tags to each word in a text to understand the grammatical structure. Useful for syntactic analysis.
    - **Example**: Identifying all nouns and verbs in a sentence to understand its structure.
- **Machine Learning Approaches**:
  - **Text Classification**: Using algorithms to classify text into predefined categories. Useful for tasks like spam detection, sentiment analysis, and topic categorization.
    - **Example**: Classifying movie reviews as positive or negative using a trained machine learning model.
  - **Named Entity Recognition (NER)**: Identifying and classifying named entities (e.g., persons, organizations, locations) in a text.
    - **Example**: Extracting names of companies and locations from news articles.
- **Deep Learning Approaches**:
  - **Recurrent Neural Networks (RNNs)**: Using RNNs and their variants (e.g., LSTM,

	GRU) for sequence modeling. Effective for tasks like language modeling and text generation. o **Example**: Generating text that mimics the style of a given author using an LSTM network. • **Transformers**: Leveraging transformer models (e.g., BERT, GPT) for various NLP tasks. These models have achieved state-of-the-art performance in many text analysis tasks. o **Example**: Using BERT for question answering, where the model finds answers to questions based on a given text. • **Hybrid Approaches**: • Combining rule-based methods with machine learning techniques to leverage the strengths of both. o **Example**: Using regular expressions to extract candidate entities from a text, followed by a machine learning model to classify and refine these entities.			
5	Briefly describe LSA feedback systems. Mention four benchmarks used by LSA to Assess the level of an explanation. Latent Semantic Analysis (LSA) is a technique in natural language processing that analyzes relationships between a set of documents and the terms they contain by producing a set of concepts related to the documents and terms. LSA feedback systems leverage this technique to provide feedback on textual content, such as student essays, explanations, or responses, by assessing the semantic content and coherence of the text. The system compares the semantic similarity between a student's text and a set of reference texts or ideal answers to provide meaningful feedback. **How LSA Feedback Systems Work** 1. **Text Representation**: Convert the text into a term-document matrix, where rows represent terms and columns represent documents. 2. **Dimensionality Reduction**: Apply Singular Value Decomposition (SVD) to reduce the dimensions of the matrix, capturing the most significant patterns in the data. 3. **Semantic Space**: Represent both the student's text and the reference texts in the same reduced-dimensional semantic space. 4. **Similarity Measurement**: Compute the cosine similarity between the student's text and reference texts to determine how closely they match in terms of semantic content. 5. **Feedback Generation**: Provide feedback based on the similarity scores, highlighting areas of strength and suggestions for improvement. **Four Benchmarks Used by LSA to Assess the Level of an Explanation** 1. **Cosine Similarity to Reference Texts**: o Measure the cosine similarity between the student's explanation and a set of reference explanations. A higher similarity score indicates that the student's text closely matches the content and context of the reference	[10] Desc-5 Examples-2	CO4	L2

texts.

2. **Coverage of Key Concepts**:
   - Assess whether the student's explanation includes key concepts and terms that are present in the reference texts. LSA can identify whether important ideas are missing or if unnecessary information is included.
3. **Coherence and Structure**:
   - Evaluate the coherence of the student's explanation by analyzing the overall structure and flow of the text. LSA can help identify if the explanation follows a logical sequence and maintains thematic consistency.
4. **Comparison with Exemplars**:
   - Compare the student's explanation with high-quality exemplar texts. By calculating the similarity scores to these exemplars, LSA can provide feedback on how closely the student's explanation aligns with well-written examples.

Example Usage in Education

In an educational setting, LSA feedback systems can be used to automatically grade essays or provide formative feedback to students. For instance, a student's essay on climate change could be compared to a set of high-quality essays on the same topic. The system would analyze the semantic content, check for the presence of key concepts like "global warming," "carbon emissions," and "renewable energy," and provide feedback on areas where the student's essay deviates from the ideal explanations.

```
FUNCTION LSA_Feedback_System(student_text, reference_texts, exemplar_texts):


    # Step 1: Preprocessing

    student_terms = preprocess_text(student_text)

    reference_terms = [preprocess_text(ref) for ref in reference_texts]

    exemplar_terms = [preprocess_text(exemplar) for exemplar in exemplar_texts]


    # Step 2: Construct Term-Document Matrix

    term_document_matrix = create_term_document_matrix(student_terms,
reference_terms + exemplar_terms)


    # Step 3: Apply Singular Value Decomposition (SVD)

    U, S, Vt = apply_SVD(term_document_matrix)


    # Step 4: Represent Texts in Semantic Space
```

```
    student_vector = project_into_semantic_space(student_terms, U, S)

    reference_vectors = [project_into_semantic_space(ref, U, S) for ref in reference_terms]

    exemplar_vectors = [project_into_semantic_space(exemplar, U, S) for exemplar in exemplar_terms]


    # Step 5: Calculate Cosine Similarities

    reference_similarities = [cosine_similarity(student_vector, ref_vector) for ref_vecto in reference_vectors]

    exemplar_similarities = [cosine_similarity(student_vector, exemplar_vector) for exemplar_vector in exemplar_vectors]


    # Step 6: Evaluate Key Concept Coverage

    key_concepts = extract_key_concepts(reference_terms)

    coverage_score = evaluate_coverage(student_terms, key_concepts)


    # Step 7: Evaluate Coherence and Structure (Optional: Simple Heuristic)

    coherence_score = evaluate_coherence(student_text)


    # Step 8: Generate Feedback

    feedback = []

    feedback.append("Similarity to Reference Texts: " + average(reference_similarities

    feedback.append("Similarity to Exemplars: " + average(exemplar_similarities))

    feedback.append("Coverage of Key Concepts: " + coverage_score)

    feedback.append("Coherence and Structure: " + coherence_score)


    RETURN feedback
```

```
FUNCTION preprocess_text(text):
    # Tokenize, convert to lowercase, and remove punctuation
    tokens = tokenize(text)
    tokens = to_lowercase(tokens)
    tokens = remove_punctuation(tokens)
    RETURN tokens

FUNCTION create_term_document_matrix(student_terms, all_terms):
    # Create a matrix where rows are terms and columns are documents
    matrix = initialize_matrix(student_terms, all_terms)
    RETURN matrix

FUNCTION apply_SVD(matrix):
    # Apply Singular Value Decomposition
    U, S, Vt = svd(matrix)
    RETURN U, S, Vt

FUNCTION project_into_semantic_space(terms, U, S):
    # Project terms into the reduced-dimensional semantic space
    vector = project(terms, U, S)
    RETURN vector

FUNCTION cosine_similarity(vector1, vector2):
    # Calculate cosine similarity between two vectors
    similarity = dot_product(vector1, vector2) / (magnitude(vector1) *
magnitude(vector2))
    RETURN similarity
```

```
FUNCTION extract_key_concepts(reference_terms):
    # Extract key concepts from reference terms
    key_concepts = identify_key_concepts(reference_terms)
    RETURN key_concepts


FUNCTION evaluate_coverage(student_terms, key_concepts):
    # Evaluate the coverage of key concepts in the student terms
    coverage = calculate_coverage(student_terms, key_concepts)
    RETURN coverage


FUNCTION evaluate_coherence(text):
    # Simple heuristic to evaluate coherence (e.g., sentence structure)
    coherence = check_coherence(text)
    RETURN coherence


FUNCTION average(similarities):
    # Calculate average similarity score
    avg_score = sum(similarities) / len(similarities)
    RETURN avg_score


# Example Usage
student_text = "The cat chased the mouse."
reference_texts = ["A cat is chasing a mouse.", "The mouse was chased by the cat."]
exemplar_texts = ["Cats often chase mice.", "In many stories, cats chase mice."]
feedback = LSA_Feedback_System(student_text, reference_texts, exemplar_texts)
```

FOR message IN feedback:

PRINT message

- **Preprocessing**: Tokenizes the input text, converts it to lowercase, and removes punctuation.
- **Term-Document Matrix**: Constructs a matrix representing the term frequencies in the documents.
- **SVD Application**: Applies Singular Value Decomposition (SVD) to reduce the dimensions of the term-document matrix.
- **Semantic Space Projection**: Projects the terms into the reduced-dimensional semantic space.
- **Cosine Similarities**: Calculates cosine similarity between the student's vector and both reference and exemplar vectors.
- **Key Concept Coverage**: Evaluates if the student's text covers key concepts present in the reference texts.
- **Coherence and Structure**: Uses a simple heuristic to evaluate the coherence and structure of the student's text.
- **Feedback Generation**: Generates feedback based on similarity scores, key concept coverage, and coherence.

6

a.Explain in detail the high-level representation approaches in text mining
b. Explain document separation as a sequence mapping problem

[10]

a-dec & examples-5

CO4 L2

High-Level Representation Approaches in Text Mining

Text mining involves the extraction of meaningful information from text. To do this effectively, various high-level representation approaches are used to convert unstructured text into structured forms that can be analyzed. Here are some prominent high-level representation approaches in text mining:

1. **Bag-of-Words (BoW)**:
   - **Description**: BoW represents a text as a collection of its words, disregarding grammar and word order but keeping multiplicity.
   - **Example**: For the sentences "The cat sat on the mat" and "The dog sat on the mat," BoW representation would count the frequency of each word in a fixed vocabulary.
   - **Advantages**: Simple and easy to implement; useful for tasks like document classification.
   - **Disadvantages**: Ignores the order and context of words, which can be crucial for understanding meaning.
2. **TF-IDF (Term Frequency-Inverse Document Frequency)**:
   - **Description**: Enhances the BoW model by weighing the frequency of a word in a document against its frequency across all documents.
   - **Example**: Words that appear frequently in one document but not in many others are given higher weights.
   - **Advantages**: Reduces the impact of common words that are less informative.
   - **Disadvantages**: Still ignores the context and sequence of words.
3. **Word Embeddings**:
   - **Description**: Uses dense vector representations for words, capturing semantic meanings based on context.
   - **Examples**: Word2Vec, GloVe, FastText.
   - **Advantages**: Captures semantic relationships between words (e.g., "king" is

to "queen" as "man" is to "woman").
  - **Disadvantages**: Requires significant computational resources for training; may capture biases present in training data.
4. **Document Embeddings**:
  - **Description**: Extends word embeddings to whole documents, capturing the overall meaning.
  - **Examples**: Doc2Vec, InferSent, Universal Sentence Encoder.
  - **Advantages**: Useful for tasks like document classification, clustering, and sentiment analysis.
  - **Disadvantages**: More complex to train than word embeddings.
5. **Topic Modeling**:
  - **Description**: Identifies topics present in a collection of documents.
  - **Examples**: Latent Dirichlet Allocation (LDA), Non-negative Matrix Factorization (NMF).
  - **Advantages**: Helps in understanding the main themes and topics within large corpora.
  - **Disadvantages**: Requires careful tuning and interpretation; topics may not always be easily interpretable.
6. **Latent Semantic Analysis (LSA)**:
  - **Description**: Reduces dimensions of the term-document matrix using Singular Value Decomposition (SVD), capturing latent semantic structures.
  - **Advantages**: Captures the underlying relationships between terms and documents.
  - **Disadvantages**: Can be computationally expensive; may not handle polysemy (multiple meanings) well.
7. **Transformers and Contextual Representations**:
  - **Description**: Uses transformer models to create contextualized word representations, where the meaning of a word is determined by its context in the sentence.
  - **Examples**: BERT, GPT, RoBERTa.
  - **Advantages**: State-of-the-art performance in many NLP tasks; captures deep semantic meaning and context.
  - **Disadvantages**: Highly computationally intensive; requires large datasets for training.

Document Separation as a Sequence Mapping Problem

b-Dec& psuedocode -5

Document separation involves dividing a continuous stream of text into distinct documents. This can be approached as a sequence mapping problem, where the task is to map a sequence of input text into sequences representing individual documents.

Steps in Document Separation as a Sequence Mapping Problem

1. **Input Sequence**:
  - The input is a continuous stream of text, such as a long transcript, email thread, or concatenated document file.
2. **Feature Extraction**:
  - Extract features that help in identifying boundaries between documents. Features can include:
    - Textual markers (e.g., titles, headers, signatures).
    - Metadata (e.g., timestamps, authorship information).
    - Content features (e.g., abrupt topic changes, specific keywords).
3. **Sequence Labeling**:
  - Treat the problem as a sequence labeling task, where each token (or

character) in the input sequence is labeled as either part of a document or a boundary.
    - Labels can be binary (boundary/non-boundary) or categorical (types of boundaries).
4. **Model Selection**:
    - Use models that are effective in sequence mapping and labeling, such as:
        - **Hidden Markov Models (HMMs)**: Probabilistic models that capture the likelihood of transitions between states (e.g., from inside a document to a boundary).
        - **Conditional Random Fields (CRFs)**: Discriminative models that consider the entire sequence for labeling, effective for capturing context.
        - **Recurrent Neural Networks (RNNs)**: Neural networks designed for sequence data, capturing dependencies over time.
        - **Transformers**: Models that handle long-range dependencies and context, suitable for processing long texts.
5. **Training**:
    - Train the chosen model on annotated data, where boundaries between documents are labeled. The model learns to recognize patterns indicative of document boundaries.
6. **Prediction**:
    - Apply the trained model to new, unlabeled sequences of text. The model predicts boundaries, effectively splitting the text into distinct documents.
7. **Post-processing**:
    - Refine the predicted boundaries based on additional rules or heuristics to ensure logical separation of documents.

```
Subject: Meeting Agenda
Hi team,
Please find the agenda for our meeting attached.

Best,
John

---Original Message---
Subject: Re: Project Update
Hi John,
Thanks for the update.

Best,
Alice
```

1. **Feature Extraction**:
    - Identify features like "Subject:", "---Original Message---", "Best," as potential boundaries.
2. **Sequence Labeling**:
    - Label tokens or characters as document parts or boundaries based on these features.
3. **Model**:
    - Use a CRF model trained on similar email threads to predict the boundaries.
4. **Prediction and Post-processing**:
    - Apply the model to label and split the email thread into individual emails.

This approach ensures accurate and automated document separation, which is crucial for

	organizing, indexing, and retrieving information from large, unstructured text streams.			

CI CCI HOD-AIML