PROGRAM 1:

Explore pre-trained word vectors. Explore word relationships using vector arithmetic. Perform arithmetic operations and analyze results.

```
import gensim.downloader as api

# Load pre-trained Word2Vec model (Google News)
print("Loading model... (This may take a while)") model =
api.load("word2vec-google-news-300") print("Model
loaded!")

# Function to find similar words
def find_similar(word):
    try:
        similar_words = model.most_similar(word)
        print(f"\nWords similar to '{word}':")
        for w, score in similar_words[:5]:  # Show top 5
            print(f"{w}: {score:.4f}")
    except KeyError:
        print(f"'{word}' not found in the vocabulary.")

# Function to perform word arithmetic
def word_arithmetic(word1, word2, word3): try:
        result = model.most_similar(positive=[word1, word2], negative=[word3]) print(f"\n'{word1}' -
        '{word3}' + '{word2}' = '{result[0][0]}' (Most similar
word)")
    except KeyError as e: print(f"Error: {e}")

# Function to check similarity between two words
def check_similarity(word1, word2): try:
        similarity = model.similarity(word1, word2)
        print(f"\nSimilarity between '{word1}' and '{word2}': {similarity:.4f}")
    except KeyError as e: print(f"Error: {e}")

# Function to find the odd one out
def odd_one_out(words):
    try:
        odd = model.doesnt_match(words) print(f"\nOdd one out
        from {words}: {odd}")
    except KeyError as e: print(f"Error: {e}")
# Run the functions
find_similar("king")
word_arithmetic("king", "woman", "man")  # Expected output: "queen"
check_similarity("king",  "queen")
odd_one_out(["apple", "banana", "grape", "car"])  # "car" should be the odd one
```

OUTPUT

```
Loading model... (This may take a while) Model
loaded!

Words similar to 'king':
kings: 0.7138
queen: 0.6511
monarch: 0.6413
crown_prince: 0.6204
prince: 0.6160

'king' - 'man' + 'woman' = 'queen' (Most similar word)

Similarity between 'king' and 'queen': 0.6511

Odd one out from ['apple', 'banana', 'grape', 'car']: car
```

Viva Questions & Answers

1. What is Word2Vec?
   Word2Vec is a shallow neural network model that learns dense vector representations of words by predicting context words (CBOW) or target words (Skip-gram).
2. What is meant by pre-trained embeddings?
   Pre-trained embeddings are word vectors trained on large corpora and reused without retraining.
3. What is the dimensionality of the Google News Word2Vec model?
   300 dimensions.
4. What does most_similar() compute?
   It returns words with highest cosine similarity to the input word vector.
5. What is cosine similarity?
   A metric measuring the cosine of the angle between two vectors.
6. Explain vector arithmetic in embeddings.
   Semantic relationships are encoded linearly; e.g.,
   king - man + woman ≈ queen.
7. Why does word arithmetic work?
   Because semantic relationships are encoded as vector offsets.
8. What does similarity() function return?
   Cosine similarity score between two words.
9. What does doesnt_match() compute?
   Identifies the word whose vector is least similar to others.
10. What is a limitation of Word2Vec?
    It generates static embeddings—each word has only one vector regardless of context.

References

- Mikolov et al., 2013. *Efficient Estimation of Word Representations in Vector Space.*
- Mikolov et al., 2013. *Distributed Representations of Words and Phrases.*
- Gensim Documentation – Word2Vec
- Jurafsky & Martin, *Speech and Language Processing*

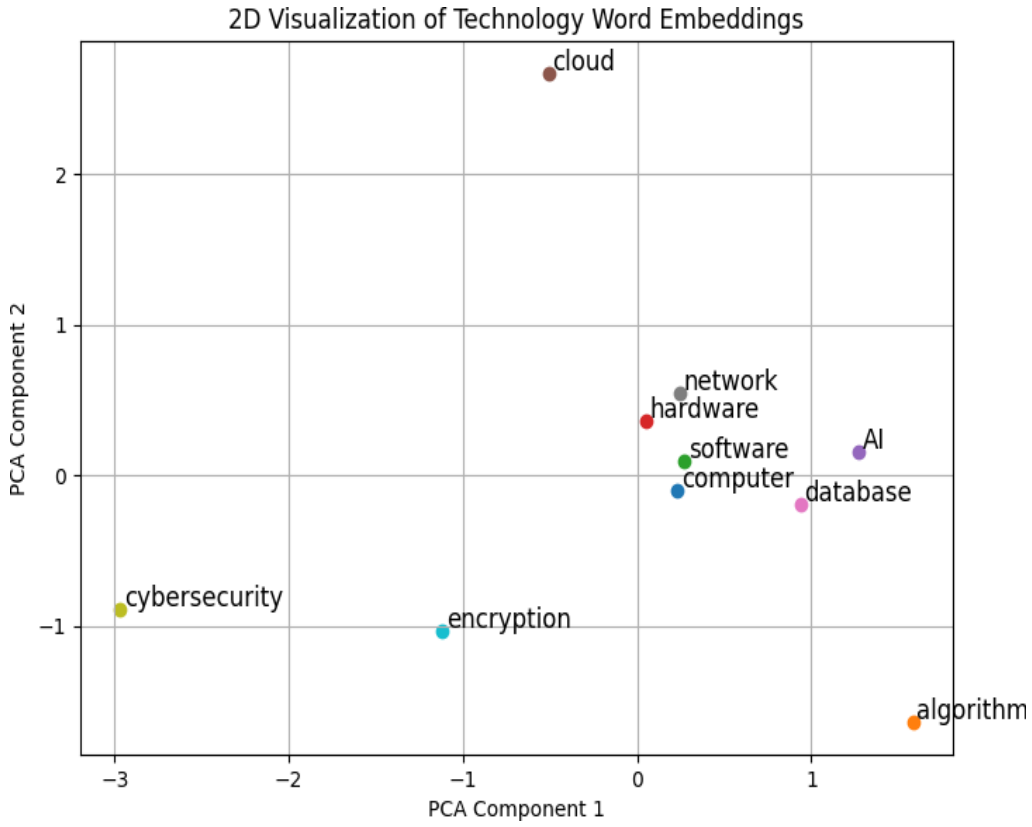
PROGRAM -2

Use dimensionality reduction (e.g., PCA or t-SNE) to visualize word embeddings for Q 1. Select 10 words from a specific domain (e.g., sports, technology) and visualize their embeddings. Analyze clusters and relationships. Generate contextually rich outputs using embeddings. Write a program to generate 5 semantically similar words for a given input.

```
import gensim.downloader as api
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Load pre-trained Word2Vec model (Google News)
print("Loading model... (This may take a while)") model =
api.load("word2vec-google-news-300") print("Model loaded!")

# Select 10 words from the Technology domain
tech_words = ["computer", "algorithm", "software", "hardware", "AI", "cloud", "database",
        "network", "cybersecurity", "encryption"]

# Get their word vectors
word_vectors = np.array([model[word] for word in tech_words])

# Perform PCA to reduce to 2D
pca = PCA(n_components=2)
reduced_vectors = pca.fit_transform(word_vectors)

# Plot the words in 2D
plt.figure(figsize=(8,6))
for word, (x, y) in zip(tech_words, reduced_vectors):
   plt.scatter(x, y)
   plt.text(x+0.02, y+0.02, word, fontsize=12)
```

```
plt.title("2D Visualization of Technology Word Embeddings") plt.xlabel("PCA Component
1")
plt.ylabel("PCA Component 2") plt.grid()
plt.show()

# Function to find 5 similar words
def find_similar_words(word):
    try:
        similar_words = model.most_similar(word, topn=5) print(f"\n5
        words similar to '{word}':")
        for w, score in similar_words: print(f"{w}:
            {score:.4f}")
    except KeyError:
        print(f"'{word}' not found in the vocabulary.")

# Test with an input word
find_similar_words("AI")
```

OUTPUT



Loading model... (This may take a while) Model loaded!

5 words similar to 'AI': Steven_Spielberg_Artificial_Intelligence: 0.5576

Index_MDE_##/###/####: 0.5415

Enemy_AI: 0.5256

Ace_Combat_Zero: 0.5227

DOA4: 0.5183

Viva Questions & Answers

1. What is PCA?
   Principal Component Analysis is a linear dimensionality reduction technique maximizing variance.
2. Why reduce dimensions for visualization?
   Human visualization is limited to 2D or 3D.
3. How does PCA work mathematically?
   By computing eigenvectors of the covariance matrix.
4. Why are embeddings high-dimensional?
   To capture rich semantic relationships.
5. What is t-SNE?
   A nonlinear dimensionality reduction technique preserving local structure.
6. Difference between PCA and t-SNE?
   PCA is linear and variance-based; t-SNE preserves neighborhood structure.
7. Why did “AI” produce unexpected similar words?
   Abbreviations may appear in varied contexts in training corpus.
8. What does clustering in embedding space indicate?
   Semantic similarity.
9. What is explained variance in PCA?
   Proportion of total variance captured by principal components.
10. Why is visualization useful in NLP?
    To interpret semantic groupings and relationships.

References

- Jolliffe, *Principal Component Analysis*
- van der Maaten & Hinton, 2008. *Visualizing Data using t-SNE*
- Gensim Documentation
- Scikit-learn Documentation

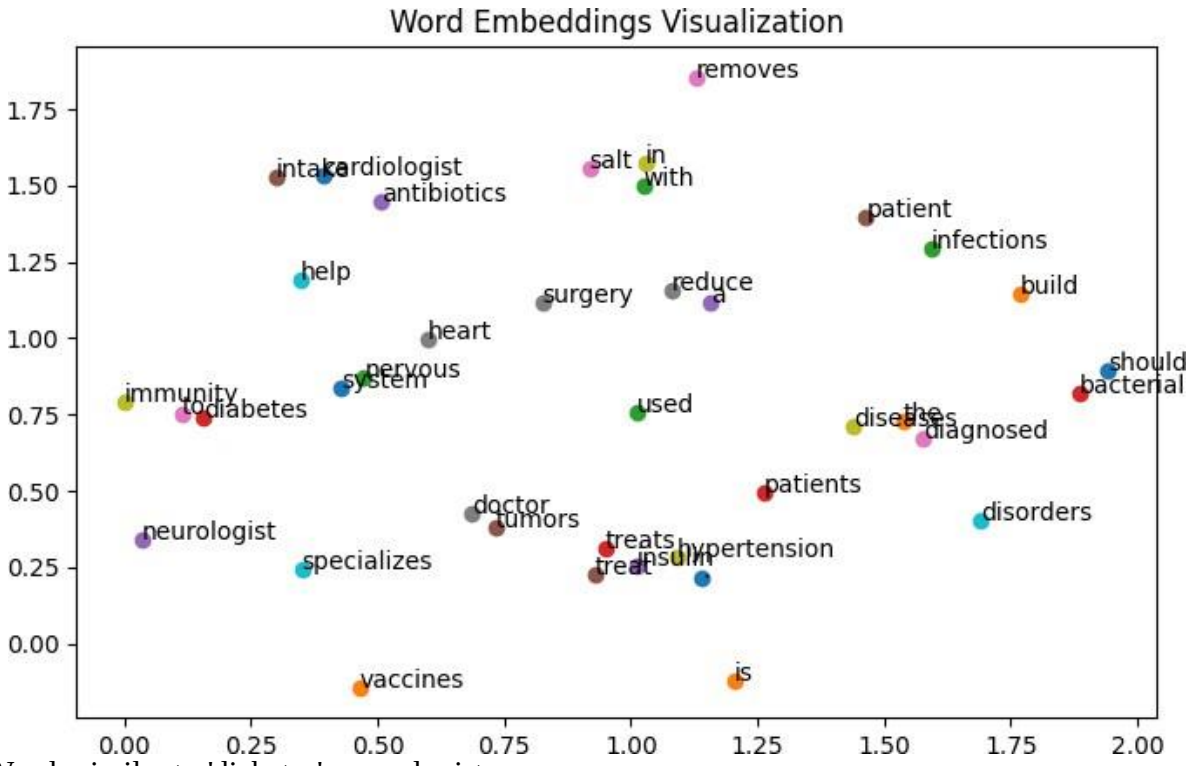
PROGRAM -3

Train a custom Word2Vec model on a small dataset. Train embeddings on a domain-specific corpus (e.g., legal, medical) and analyze how embeddings capture domain-specific semantics.

```
import gensim
from gensim.models import Word2Vec
import nltk
from nltk.tokenize import word_tokenize
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

# Medical corpus
medical_corpus = [
    "The doctor diagnosed the patient with diabetes.", "Insulin is
    used to treat diabetes.",
    "A cardiologist specializes in heart diseases.",
    "Patients with hypertension should reduce salt intake.", "Antibiotics treat
    bacterial infections.",
    "Vaccines help build immunity.", "Surgery
    removes tumors.",
    "A neurologist treats nervous system disorders.",
]

# Tokenization
nltk.download('punkt')
tokenized_corpus = [word_tokenize(sentence.lower()) for sentence in
medical_corpus]

# Train Word2Vec model
model = Word2Vec(sentences=tokenized_corpus, vector_size=50, window=5, min_count=1, workers=4)

# Save model
model.save('medical_word2vec.model')

def plot_embeddings(model):
    words = list(model.wv.index_to_key) word_vectors =
    model.wv[words]

    # Reduce dimensions using t-SNE
    tsne = TSNE(n_components=2, random_state=42)
    reduced_vectors = tsne.fit_transform(word_vectors)

    plt.figure(figsize=(8, 5))
    for i, word in enumerate(words): x, y =
        reduced_vectors[i] plt.scatter(x, y)
```

```
        plt.annotate(word, (x, y), fontsize=10)

    plt.title("Word Embeddings Visualization") plt.show()

# Plot word embeddings
plot_embeddings(model)

# Find similar words
def find_similar_words(word):
    if word in model.wv:
        similar_words = model.wv.most_similar(word, topn=5) print(f"Words
        similar to '{word}':")
        for similar, score in similar_words: print(f"{similar}
            (Similarity: {score:.2f})")
    else:
        print(f"'{word}' not found in vocabulary")

# Example usage
find_similar_words("diabetes")
```

OUTPUT



Words similar to 'diabetes': neurologist
(Similarity: 0.23)
help (Similarity: 0.20)
used (Similarity: 0.19)
vaccines (Similarity: 0.17)
surgery (Similarity: 0.13)

Viva Questions & Answers

1. Why train a domain-specific model?
   To capture specialized terminology semantics.
2. What is vector_size?
   Dimensionality of word embeddings.
3. What is window parameter?
   Context window size.
4. What is min_count?
   Minimum frequency threshold.
5. Why are similarities low in small corpus?
   Limited training data reduces semantic richness.
6. What is Skip-gram vs CBOW?
   Skip-gram predicts context; CBOW predicts target word.
7. Why use t-SNE here?
   To visualize nonlinear relationships.
8. What is tokenization?
   Splitting text into words/tokens.
9. Why lowercase text?
   To reduce vocabulary size.
10. Limitation of small datasets?
    Poor generalization and unstable embeddings.

References

- Mikolov et al., 2013
- Gensim Word2Vec Documentation
- Jurafsky & Martin
- NLTK Documentation

PROGRAM -4

Use word embeddings to improve prompts for Generative AI model. Retrieve similar words using word embeddings. Use the similar words to enrich a GenAI prompt. Use the AI model to generate responses for the original and enriched prompts. Compare the outputs in terms of detail and relevance.

Follow the steps to run the program if error occurs

step1 : - Update the opt tree
python -m pip install --upgrade "optree>=0.13.0"

step 2: install tf keras library pip
install tf-keras

step3:- Fix TensorFlow one DNN Warnings run the program in VSCODE termninal set

TF_ENABLE_ONEDNN_OPTS=0 # Windows Command Prompt

Step 4 :- check the required libraries
pip install --upgrade transformers gensim torch tensorflow optree

Now run the program

```
import gensim.downloader as api
from transformers import pipeline

# Load embedding model
embedding_model = api.load("glove-wiki-gigaword-100") original_prompt =

"Describe the beautiful landscapes during sunset."

def enrich_prompt(prompt, embedding_model, n=5): words
   = prompt.split()
   enriched_prompt = []

   for word in words: word_lower =
      word.lower()

      if word_lower in embedding_model:
         similar_words = embedding_model.most_similar(word_lower, topn=n) similar_word_list = [w[0]
         for w in similar_words]
         enriched_prompt.append(" ".join(similar_word_list)) # Join similar words as a phrase else:
         enriched_prompt.append(word) # Keep the word as is if not found


   return " ".join(enriched_prompt)
```

```
enriched_prompt = enrich_prompt(original_prompt, embedding_model)

# Load text generation model
generator = pipeline("text-generation", model="gpt2")

# Generate responses
original_response = generator(original_prompt, max_length=50, num_return_sequences=1)
enriched_response = generator(enriched_prompt, max_length=50, num_return_sequences=1)

# Print results
print("Original prompt response")
print(original_response[0]['generated_text'])

print("\nEnriched prompt response")
print(enriched_response[0]['generated_text'])
```

OUTPUT

Original prompt response
Describe the beautiful landscapes during sunset. View the entire project » View more photos

View slideshow

View gallery

Enriched prompt response
explain describing distinguish understand define this part one of same lovely gorgeous wonderful charming magnificent landscape seascapes cityscapes scenery paintings after early since following days sunset. cityscape paintings after all great of beautiful seascapes and also a vast beautiful

Viva Questions & Answers

1. What is GloVe?
   Global Vectors for Word Representation based on global co-occurrence statistics.
2. Why enrich prompts?
   To increase contextual diversity and semantic richness.
3. What is GPT-2?
   Transformer-based autoregressive language model.
4. Difference between embeddings and LLMs?
   Embeddings represent words; LLMs generate sequences.
5. Why did enriched prompt produce longer output?
   Increased semantic input tokens influence generation.
6. What is tokenization in transformers?
   Splitting text into subword tokens.
7. What is pipeline() in transformers?
   High-level API for model tasks.
8. Why may enrichment degrade coherence?
   Excess synonyms reduce contextual clarity.
9. What is contextual generation?

Text generated based on preceding tokens.

10. What is attention mechanism?

    Weighted focus on input tokens during generation.

References

- Pennington et al., 2014. *GloVe*
- Radford et al., 2019. *GPT-2*
- Hugging Face Transformers Documentation

PROGRAM -5

Use word embeddings to create meaningful sentences for creative tasks. Retrieve similar words for a seed word. Create a sentence or story using these words as a starting point. Write a program that: Takes a seed word. Generates similar words. Constructs a short paragraph using these words

```
import random
import gensim.downloader as api

# Load a pre-trained word embedding model
model = api.load("glove-wiki-gigaword-50")  # 50D GloVe embeddings

def get_similar_words(seed_word, top_n=5): """Retrieve similar
    words for the given seed word."""  try:
        similar_words = [word for word, _ in model.most_similar(seed_word,
topn=top_n)]
        return similar_words
    except KeyError:
        return [ ]

def create_paragraph(seed_word):
    """Generate a short paragraph using the seed word and its similar words."""
    similar_words = get_similar_words(seed_word)

    if not similar_words:
        return f"Could not find similar words for '{seed_word}'. Try another word!"

    # Create a simple paragraph
    paragraph = (
        f"Once upon a time, a {seed_word} embarked on a journey. Along the way, it encountered "
        f"a {random.choice(similar_words)}, which led it to a hidden
{random.choice(similar_words)}.  "
        f"Despite the challenges, it found {random.choice(similar_words)} and embraced the "
        f"adventure with {random.choice(similar_words)}. In the end, the journey was a tale of "
        f"{random.choice(similar_words)} and discovery."
    )

    return paragraph
# Example usage
seed_word = input("Enter a seed word: ").strip().lower()
print("\nGenerated Story:\n") print(create_paragraph(seed_word))
```

OUTPUT

Enter a seed word: adventure Generated Story:

Once upon a time, a adventure embarked on a journey. Along the way, it encountered a adventures, which led it to a hidden adventures. Despite the challenges, it found romance and embraced the adventure with mystery. In the end, the journey was a tale of mystery and discovery.

Viva Questions & Answers

1. How are similar words retrieved?
   Using cosine similarity.
2. Why randomness is used?
   To increase variability in output.
3. What is a seed word?
   Initial word guiding semantic expansion.
4. Why grammatical errors occurred?
   No syntactic validation in generation.
5. Difference between rule-based and neural generation?
   Rule-based follows grammar rules; neural learns patterns.
6. hy embeddings alone cannot ensure coherence?
   They lack sentence-level modeling.
7. What improves story quality?
   Transformer-based generative models.
8. What is semantic neighborhood?
   Words close in embedding space.
9. Why 50D embeddings used?
   Lightweight and faster.
10. Limitation of static embeddings?
    No context adaptation.

References

- GloVe Paper (2014)
- Gensim Documentation
- Jurafsky & Martin

PROGRAM -6

Use a pre-trained Hugging Face model to analyze sentiment in text. Assume a real-world application, Load the sentiment analysis pipeline. Analyze the sentiment by giving sentences to input.

```
from transformers import pipeline

# Load the sentiment analysis pipeline
sentiment_analyzer = pipeline("sentiment-analysis")

def analyze_sentiment(text):
    """Analyze sentiment of the input text using Hugging Face pipeline."""
    result = sentiment_analyzer(text) label =
    result[0]['label']
    score = result[0]['score']

    return f"Sentiment: {label} (Confidence: {score:.2f})"

# Example usage while
True:
    user_input = input("Enter a sentence for sentiment analysis (or 'exit' to quit): ").strip()
    if user_input.lower() == 'exit':
        break
    print(analyze_sentiment(user_input))
```

OUTPUT

```
Device set to use CPU
Enter a sentence for sentiment analysis (or 'exit' to quit): I love this product! It's amazing
Sentiment: POSITIVE (Confidence: 1.00)
Enter a sentence for sentiment analysis (or 'exit' to quit): This is the worst experience ever
Sentiment: NEGATIVE (Confidence: 1.00)
Enter a sentence for sentiment analysis (or 'exit' to quit): The service was okay, nothing special
Sentiment: NEGATIVE (Confidence: 0.99)
```

Viva Questions & Answers

1. What model is used in default pipeline?
   DistilBERT fine-tuned on SST-2.
2. What is sentiment analysis?
   Classification of text polarity.
3. What does confidence score represent?
   Softmax probability.
4. Why neutral sentence classified negative?
   Binary classifier (positive/negative only).
5. What is fine-tuning?
   Training pre-trained model on specific task dataset.

6. What is transformer architecture?
   Attention-based sequence model.
7. Why CPU used?
   No GPU available.
8. What is classification head?
   Final dense layer for label prediction.
9. How is sentiment predicted?
   Based on contextual token embeddings.
10. Limitation?
    Domain sensitivity.

References

- Devlin et al., 2018. *BERT*
- Hugging Face Documentation
- Vaswani et al., 2017. *Attention is All You Need*

PROGRAM -7

Summarize long texts using a pre-trained summarization model using Hugging face model. Load the summarization pipeline. Take a passage as input and obtain the summarized text.

```
from transformers import pipeline # Load

the summarization model
summarizer = pipeline("summarization", model="facebook/bart-large-cnn")

# Take user input for the text passage
text = input("Enter the text you want to summarize:\n")

# Summarize the text
summary = summarizer(text, max_length=100, min_length=30, do_sample=False)

# Print the summarized text print("\nSummarized Text:")
print(summary[0]['summary_text'])
```

OUTPUT

Enter the text you want to summarize: The Indian Penal Code (IPC) is the official criminal code of India. It was drafted in 1860 by the first law commission of India, under the chairmanship of Thomas Babington Macaulay. The IPC covers a wide range of offenses, including crimes against the state, public tranquillity, human body, property, and morality. It provides detailed provisions for punishment, ranging from fines to imprisonment and even the death penalty for severe crimes. Over the years, various amendments have been made to the IPC to accommodate evolving legal and social requirements. The code serves as the foundation for criminal law in India, ensuring justice and order in society.

Summarized Text:
The Indian Penal Code (IPC) is the official criminal code of India. It was drafted in 1860 by the first law commission of India, under the chairmanship of Thomas Babington Macaulay. The code covers a wide range of offenses, including crimes against the state, public tranquillity, human body, property, and morality.

Viva Questions & Answers

1. **What type of summarization is used?**
   Abstractive summarization.
2. **What is BART?**
   Denoising autoencoder transformers.
3. **Difference between extractive and abstractive summarization?**
   Extractive selects sentences; abstractive generates new text.
4. **What is max_length?**
   Maximum summary token length.
5. **Why deterministic output?**

do_sample=False.

6. **What is encoder-decoder architecture?**
   Encoder processes input; decoder generates output.
7. **Why BART suitable for summarization?**
   Pretrained on text reconstruction tasks.
8. **What dataset used for training?**
   CNN/DailyMail.
9. **What is beam search?**
   Decoding strategy exploring multiple paths.
10. **Limitation?**
    May hallucinate facts.

References

- Lewis et al., 2020. *BART*
- Hugging Face Documentation

PROGRAM -8

Install langchain, cohere (for key), langchain-community. Get the api key(By logging into Cohere and obtaining the cohere key). Load a text document from your google drive . Create a prompt template to display the output in a particular manner

```
from langchain.prompts import PromptTemplate from
langchain_community.llms import Cohere

# Set your Cohere API key
COHERE_API_KEY = "YOUR_COHERE_API"  # Replace with your actual API key

# Read the text file
file_path = "Artificial_Intelligence.txt"  # Replace with your file name

with open(file_path, "r", encoding="utf-8") as file: document_text
   = file.read()


print("File loaded successfully!")
```

Output 1: -

```
File loaded successfully! # Create a simple


prompt template
prompt_template = PromptTemplate(
   input_variables=["text"],
   template="Summarize the following text in a simple way:\n\n{text}"
)

# Initialize the Cohere model
llm = Cohere(cohere_api_key=COHERE_API_KEY)

# Run the text through Cohere
output = llm.invoke(prompt_template.format(text=document_text))

# Display the result print("Summary:\n", output)
```

OUTPUT 2: -

Summary:

Here's a simplified summary of the text:

Artificial Intelligence (AI) is a technology that makes machines smart. It enables them to perform tasks like humans by learning, solving problems, and making decisions. AI is used in many areas to help improve performance. Some examples of AI include machine learning, deep learning, and robotics. However, there are some concerns about AI, like ensuring privacy and fairness, and how it could impact jobs. Overall, AI can be very helpful, but it's important to use it responsibly.

Viva Questions & Answers

1. What is LangChain?
   Framework for LLM-based applications.
2. What is PromptTemplate?
   Structured prompt generator.
3. What is Cohere?
   LLM API provider.
4. Why use template?
   Standardized prompt formatting.
5. What is LLM invocation?
   Sending prompt to model API.
6. What is API key?
   Authentication credential.
7. Why summary simplified?
   Instruction in template.
8. Difference between pipeline and LangChain?
   Pipeline is task-based; LangChain orchestrates workflows.
9. What is chain?
   Sequence of model calls.
10. Security concern?
    Protect API keys.

References

- LangChain Documentation
- Cohere Documentation

PROGRAM -9

Take the Institution name as input. Use Pydantic to define the schema for the desired output and create a custom output parser. Invoke the Chain and Fetch Results. Extract the below Institution related details from Wikipedia: The founder of the Institution. When it was founded. The current branches in the institution . How many employees are working in it. A brief 4-line summary of the institution.

```
from pydantic import BaseModel import
wikipediaapi

# Define the Pydantic schema
class InstitutionDetails(BaseModel): name: str
    founder: str
    founded_year: str
    branches: str
    employees: str
    summary: str

# Wikipedia extraction function
def fetch_institution_details(institution_name: str) -> InstitutionDetails: wiki_wiki =
    wikipediaapi.Wikipedia(user_agent="MyWikipediaScraper/1.0
(contact: myemail@example.com)", language="en") page =
    wiki_wiki.page(institution_name)

    if not page.exists():
        raise ValueError("Institution page does not exist on Wikipedia")

    # Extract information (this part needs actual content parsing) summary = "
    ".join(page.summary.split(".")[:4]) + "."
    # Placeholder extraction logic founder = "Not
    Available" founded_year = "Not Available"
    branches = "Not Available" employees =
    "Not Available"

    for section in page.sections:
        if "founder" in section.title.lower(): founder =
            section.text.split(". ")[0]
        if "founded" in section.title.lower(): founded_year =
            section.text.split(". ")[0]
        if "branches" in section.title.lower(): branches =
            section.text.split(". ")[0]
        if "employees" in section.title.lower(): employees =
            section.text.split(". ")[0]

    return InstitutionDetails(
        name=institution_name, founder=founder,
```

```
        founded_year=founded_year,
        branches=branches, employees=employees,
        summary=summary
    )

# Example invocation
institution_name = input("Enter Institution Name: ") try:
    details = fetch_institution_details(institution_name) print(details.model_dump_json(indent=4))

except ValueError as e: print(str(e))
```

OUTPUT

```
Enter Institution Name:  Harvard University
{
    "name": "Harvard University", "founder":
    "Not Available", "founded_year": "Not
    Available", "branches": "Not Available",
    "employees": "Not Available",
    "summary": "Harvard University is a private Ivy League research university in Cambridge, Massachusetts, United States  Founded October 28, 1636, and named for its first benefactor, the Puritan clergyman John Harvard, it is the oldest institution of higher learning in the United States  Its influence, wealth, and rankings have made it one of the most prestigious universities in the world \nHarvard was founded and authorized by the Massachusetts General Court, the governing legislature of colonial- era Massachusetts Bay Colony."
}
```

Viva Questions & Answers

1. What is Pydantic?
   Data validation library.
2. Why schema needed?
   Structured output enforcement.
3. What is BaseModel?
   Parent class for data models.
4. What is JSON dump?
   Convert object to JSON format.
5. Why placeholders returned?
   Section parsing limited.
6. What is web scraping?
   Extracting web data programmatically.
7. Why Wikipedia API used?
   Structured access to pages.
8. What is model validation?
   Ensuring data conforms to schema.
9. Limitation?
   Section headings inconsistent.

10. Improvement?
    Use NLP-based information extraction.

References

- Pydantic Documentation
- Wikipedia API Documentation

PROGRAM 10

Build a chatbot for the Indian Penal Code. We'll start by downloading the official Indian Penal Code document, and then we'll create a chatbot that can interact with it. Users will be able to ask questions about the Indian Penal Code and have a conversation with it.

Step 1: Download the IPC PDF File
Before running the chatbot, you must download the Indian Penal Code (IPC) PDF. Use the following Python script to download it:

```
import requests

def download_ipc_pdf(url, save_path="ipc.pdf"): try:
        response = requests.get(url)
        response.raise_for_status()  with
        open(save_path, 'wb') as f:
            f.write(response.content) print(f"Downloaded IPC PDF
        to: {save_path}")
    except requests.exceptions.RequestException as e:
        print(f"Request error: {e}")
    except Exception as e: print(f"Unexpected
        error: {e}")

if __name____== "__main__":
    ipc_pdf_url ="https://www.indiacode.nic.in/bitstream/123456789/4219/1/THE- INDIAN-PENAL-CODE-1860.pdf"
    download_ipc_pdf(ipc_pdf_url)
```

Instructions:

1. **Save this script** as a .py file (e.g., download_ipc.py).
2. **Run the script** to download the IPC PDF to your local system. By default, the file will be saved as ipc.pdf.
3. **Ensure the file** is saved in the directory where you intend to run the chatbot application, or provide the appropriate path.

Step 2: Run the Chatbot
Once the IPC PDF is downloaded, you can proceed with running the chatbot. The chatbot will read from the ipc.pdf file to provide responses based on the Indian Penal Code.

```
import PyPDF2
import re import
nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize import string
from sklearn.feature_extraction.text import TfidfVectorizer from
sklearn.metrics.pairwise import cosine_similarity

def extract_text_from_pdf(pdf_path): text = ""
    try:
```

```
        with open(pdf_path, 'rb') as file: reader =
            PyPDF2.PdfReader(file) for page in
            reader.pages:
                text += page.extract_text() or "" return text
    except FileNotFoundError:
        print(f"Error: File not found at {pdf_path}") return ""
    except Exception as e:
        print(f"Error extracting text from PDF: {e}") return ""

def preprocess_text(text): if not
    text:
        return []
    text = text.lower()
    text = text.translate(str.maketrans('', '', string.punctuation)) tokens =
    word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words] return tokens

def create_index(text): index =
    {}
    try:
        section_pattern =
r"((?:CHAPTER|SECTION)\s+\w+\.?\s+.*?)(?:(?:CHAPTER|SECTION)\s+\w+\.?\s+|$
)"
        matches = re.findall(section_pattern, text, re.DOTALL | re.IGNORECASE) for match in
        matches:
            title_match = re.search(r"^(?:CHAPTER|SECTION)\s+\w+\.?\s+(.*?)(?=\n)", match,
re.DOTALL | re.IGNORECASE)
            if title_match:
                title = title_match.group(1).strip()
                content = match[title_match.end():].strip() index[title] =
                content
        return index
    except Exception as e:
        print(f"Error creating index: {e}") return {}

def get_most_relevant_section(query, index): try:
        if not index: return
            None
        sections = list(index.values()) section_titles
        = list(index.keys()) vectorizer =
        TfidfVectorizer()
        tfidf_matrix = vectorizer.fit_transform(sections + [query]) query_vector =
        tfidf_matrix[-1]
        similarities = cosine_similarity(query_vector, tfidf_matrix[:-1]).flatten() if not
        similarities.any():
            return None
        most_relevant_index = similarities.argmax() return
        section_titles[most_relevant_index]
```

```
    except Exception as e:
        print(f"Error finding relevant section: {e}") return
        None
def get_section_text(section_title, index): try:
        return index.get(section_title) except
    Exception as e:
        print(f"Error getting section text: {e}") return None

def generate_response(query, index): if not
    index:
        return "I'm sorry, I cannot process the IPC without the index. Please ensure the IPC content is
loaded."
    relevant_section_title = get_most_relevant_section(query, index) if not
    relevant_section_title:
        return "I'm sorry, I couldn't find relevant information in the IPC for your query." section_text =
    get_section_text(relevant_section_title, index)
    if not section_text:
        return "I found the relevant section, but I'm unable to retrieve the details." cleaned_text =
    re.sub(r'\n\s*\n+', '\n\n', section_text.strip())
    cleaned_text = re.sub(r'[ \t]+\n', '\n', cleaned_text) cleaned_text
    = re.sub(r'\n+', '\n', cleaned_text)
    response = f"Here's what I found in the Indian Penal Code,
**{relevant_section_title}**:\n\n{cleaned_text}" return
    response

def chatbot(index):
    print("Welcome to the Indian Penal Code Chatbot! Ask me anything about the IPC. Type 'exit' to
quit.")
    while True:
        query = input("You: ")
        if query.lower() == "exit": break
        response = generate_response(query, index) print(f"Chatbot:
        {response}\n")

if __name____== "__main__":
    pdf_path = "ipc.pdf"  # Path to your local IPC PDF ipc_text =
    extract_text_from_pdf(pdf_path)
    if ipc_text:
        ipc_index = create_index(ipc_text) if
        ipc_index:
            chatbot(ipc_index) else:
            print("Failed to create index. Chatbot cannot start.")
    else:
        print("Failed to extract text from PDF. Chatbot cannot start.")
```

OUTPUT

```
Welcome to the Indian Penal Code Chatbot! Ask me anything about the IPC. Type 'exit' to quit.
You: Murder
Chatbot: Here's what I found in the Indian Penal Code, **word "offence" includes every**:
```

act committed outside 1*[India] which, if committed in 1*[India] would be punishable under this Code.

2*[Illustration]

3***A, 4*[who is 5*[a citizen of India]], commits a murder in Uganda. He can be tried and convicted of murder in any place in 1*[India] in which he may be found.

6* * * * *

5.

Certain laws not to be affected by this Act.

7*[5. Certain laws not to be affected by this Act.--Nothing in this Act shall affect the provisions of any Act for punishing mutiny and desertion of officers, soldiers, sailors or airmen in the service of the Government of India or the provision of any special or local law.]

You: exit

Viva Questions & Answers

1. What is TF-IDF?
   Term Frequency–Inverse Document Frequency weighting scheme.
2. Why cosine similarity used?
   Measures document similarity.
3. Why preprocessing needed?
   Remove noise.
4. What are stopwords?
   High-frequency non-informative words.
5. Why section indexing?
   Faster retrieval.
6. What is vectorization?
   Converting text into numerical form.
7. Limitation of TF-IDF chatbot?
   No deep contextual understanding.
8. Why PDF extraction needed?
   Source document parsing.
9. Difference from LLM chatbot?
   Retrieval-based vs generative.
10. What improves performance?
    Using embeddings + semantic search.

References

- Salton & Buckley, 1988. TF-IDF
- Scikit-learn Documentation
- Jurafsky & Martin
- NLTK Documentation