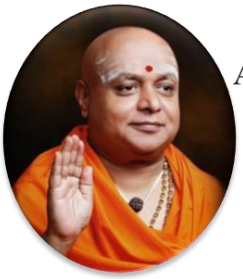
||Jai Sri Gurudev||



ADICHUNCHANAGIRI INSTITUTE OF TECHNOLOGY

CHIKKAMAGALURU

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

LAB MANUAL

SEMESTER VI

Generative AI
(BAIL657C)
(Academic Year 2024-2025)

AEC/SDC- V: Ability Enhancement Course/Skill Development Course V

[As per Choice Based Credit System (CBCS) scheme]

When all castes and creeds live together with a sense of equality and brotherhood, they will be truly happy and country strong. Collective organization of society has its rewards.

--Sri Sri Sri Dr. Balagangadharanatha Mahaswamiji

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi, Karnataka –590018

SEMESTER VI

Generative AI

(BAIL657C)

(Academic Year 2024-2025)

AEC/SDC- V: Ability Enhancement Course/Skill Development Course V

[As per Choice Based Credit System (CBCS) scheme]

ADICHUNCHANAGIRI INSTITUTE OF TECHNOLOGY

8QRX+P7W, BEEKANAHALLI RURAL CHIKKAMAGALURU, KARNATAKA 577101

(2024-2025)

Mr. Abhishek N B.E., M.Tech,.
Assistant Professor,
Dept. of IS&E,
A.I.T Chikkamagaluru,

Dr. Sampath S B.E., M.Tech., Ph.D.,
Professor & Head,
Dept. of IS&E,
A.I.T Chikkamagaluru,

DEPARMENT VISION AND MISSION

The department was started in the year 1999-2000 with an intake of 60. Department comprises of highly qualified and experienced teaching staff and research scholars.

Department is performing well by securing VTU ranks in academics. With support from the institution, the department is constantly providing career guidance for the students in order to achieve good results and it has excellent placement records.

Department regularly conducts technical talks, seminars, short-term courses, hands-on training, and workshops for students and faculty to bridge the technical gap between educational institutions and industries.

VISION

To be recognized as a centre of excellence in information technology and allied areas with quality learning and research environment

MISSION

- Provide intellectual & professional leadership in ethical and social areas pertaining to information in contemporary society.
- Advancing the state of knowledge of information studies through research and development.
- Advancing the state of knowledge of information studies through research and development.

Course objectives:

- Understand the principles and concepts behind generative AI models
- Explain the knowledge gained to implement generative models using Prompt design frameworks.
- Apply various Generative AI applications for increasing productivity.
- Develop Large Language Model-based Apps.

Course outcome:

- Develop the ability to explore and analyze word embeddings, perform vector arithmetic to investigate word relationships, visualize embeddings using dimensionality reduction techniques
- Apply prompt engineering skills to real-world scenarios, such as information retrieval, text generation.
- Utilize pre-trained Hugging Face models for real-world applications, including sentiment analysis and text summarization.
- Apply different architectures used in large language models, such as transformers, and understand their advantages and limitations

Do's

- Maintain Silence inside the Laboratory.
- Maintain Cleanliness.
- Leave your footwear outside the Laboratory.
- Enter the required details in the Log Book.
- Handle the Laboratory Instrument Carefully.
- Bring Laboratory Record and the Observation Book regularly.
- Any malfunctioning of Computer should be brought to the notice of Laboratory In-charge immediately.
- Facilities can be utilized during the free slots with the permission of Laboratory In-charge.
- Always shut down your computer from start menu and ensure the terminal/computer is logged off property.

Don'ts

- Do not delete or copy the files from the system.
- Do not displace Laboratory equipment.
- Do not shut down your computer by switching off the power directly.

These are Strictly Prohibited

- Entering the Laboratory without uniform.
- Group discussion or chatting.
- Walking around the Laboratory.
- Mobile Phone and Pen Drive.

LIST OF CONTENTS

10	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.
11	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.

EXPERIMENT-1

EXPLORE PRE-TRAINED WORD VECTORS. EXPLORE WORD RELATIONSHIPS USING VECTOR ARITHMETIC. PERFORM ARITHMETIC OPERATIONS AND ANALYZE RESULTS.

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

Load Pre-trained Word Vectors

```
from gensim.models import KeyedVectors
word_vectors = KeyedVectors.load_word2vec_format (‘C:\\Users\\Admin\\Documents\\GAI LAB\\GoogleNews-vectors-negative300.bin', binary=True)
```

Explore Word Relationships

```
result = word_vectors.most_similar(positive=['woman', 'king'], negative=['man'], topn=1)
print(result)
```

Output

result

[('queen', 0.7118193507194519)]

Perform Arithmetic Operations

```
result = word_vectors.most_similar(positive=['brother', 'woman'], negative=['man'], topn=1)
print(result)
```

Output

result

[('sister', 0.8103213906288147)]

Analyze Results

```
result = word_vectors.most_similar(positive=['doctor', 'woman'], negative=['man'], topn=1)
print(result)
```

Output

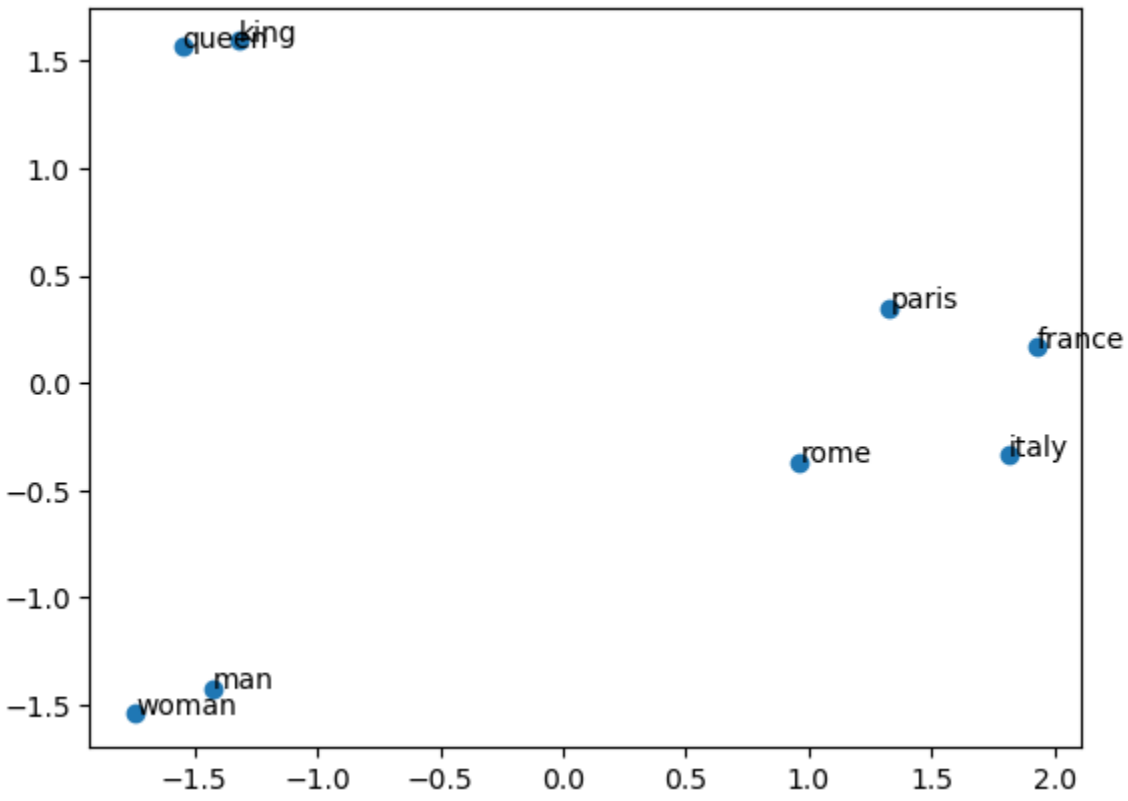
result

[('gynecologist', 0.7093892097473145)]

Visualize Word Vectors

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
words = ['king', 'queen', 'man', 'woman', 'paris', 'france', 'rome', 'italy']
vectors = [word_vectors[word] for word in words]
pca = PCA(n_components=2)
vectors_2d = pca.fit_transform(vectors)
# Plot the vectors
plt.scatter(vectors_2d[:, 0], vectors_2d[:, 1])
for i, word in enumerate(words):
    plt.annotate(word, xy=(vectors_2d[i, 0], vectors_2d[i, 1]))
plt.show()
```

Output:



Result: Explore pre-trained word vectors. Explore word relationships using vector arithmetic. Perform arithmetic operations and analyze results has been successfully executed

EXPERIMENT-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.

AIM: 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.

Load Pre-trained Word Vectors

```
from gensim.models import KeyedVectors
word_vectors = KeyedVectors.load_word2vec_format('C:\\Users\\Admin\\Documents\\AI LAB\\GoogleNews-vectors-negative300.bin', binary=True)
```

Select 10 Words from a Specific Domain

```
words = ['computer', 'software', 'hardware', 'algorithm', 'data', 'network', 'internet', 'programming', 'database', 'cloud']
```

Extract Word Vectors

```
import numpy as np
vectors = np.array([word_vectors[word] for word in words])
```

Apply Dimensionality Reduction (PCA or t-SNE)

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
pca = PCA(n_components=2)
vectors_2d = pca.fit_transform(vectors)
plt.figure(figsize=(10, 8))
plt.scatter(vectors_2d[:, 0], vectors_2d[:, 1])
for i, word in enumerate(words):
```

```
    plt.annotate(word, xy=(vectors_2d[i, 0], vectors_2d[i, 1]), fontsize=12)
plt.title('PCA Visualization of Technology Word Embeddings')
plt.show ( )
```

Analyze Clusters and Relationships Words like "computer", "hardware", and "software" may cluster together because they are closely related in the technology domain.

Generating Semantically Similar Words

```
def find_similar_words(word, topn=5):
    try:
        similar_words = word_vectors.most_similar(word, topn=topn)
        print(f"Words similar to '{word}':")
        for word, similarity in similar_words:
            print(f"{word}: {similarity:.2f}")
    except KeyError:
        print(f"'{word}' is not in the vocabulary.")
# Example usage
find_similar_words('computer')
```

Words similar to 'computer':
computers: 0.80
laptop: 0.66
laptop_computer: 0.65
Computer: 0.65
com_puter: 0.61

Test with Different Words

```
find_similar_words('algorithm')
find_similar_words('cloud')
find_similar_words('computer')
find_similar_words('sports')
find_similar_words('Technology')
```

```
Words similar to 'algorithm':
algorithms: 0.84
mathematical_algorithm: 0.65
algorithmically: 0.64
Algorithm: 0.64
algorithmic: 0.63
Words similar to 'cloud':
clouds: 0.76
Cloud: 0.60
Abu_Risha_assassination: 0.60
Carefully_cautiously: 0.59
cloud_computing: 0.56
Words similar to 'computer':
computers: 0.80
laptop: 0.66
laptop_computer: 0.65
Computer: 0.65
com_puter: 0.61
Words similar to 'sports':
sport: 0.69
sporting: 0.64
Sports: 0.63
DeVillers_reports: 0.61
athletics: 0.61
Words similar to 'Technology':
Technolgy: 0.67
Techonology: 0.64
Techology: 0.62
Controlled_Deployment: 0.60
excluding_Arques: 0.59
```

Result:

EXPERIMENT-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.

AIM: 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.

Create a Text file

The plaintiff filed a lawsuit against the defendant.
The court ruled in favor of the claimant. The contract was terminated due to breach of agreement.
The judge issued a subpoena for the witness. The attorney presented evidence to the jury.
The defendant pleaded not guilty to the charges.
The court dismissed the case due to lack of evidence.
The parties agreed to a settlement outside of court.
The plaintiff sought damages for emotional distress.
The judge granted a motion to dismiss.

Train a Custom Word2Vec Model

```
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess
with open('legal_corpus.txt', 'r') as file:
    sentences = file.readlines()
processed_sentences = [simple_preprocess(sentence) for sentence in sentences]
model = Word2Vec(sentences=processed_sentences, vector_size=100, window=5, min_count=1, workers=4)
model.save("legal_word2vec.model")
```

Analyze Domain-Specific Semantics

```
similar_words = model.wv.most_similar('lawsuit', topn=5)
print("Words similar to 'lawsuit':", similar_words)
```

Words similar to 'lawsuit': [('witness', 0.19192539155483246), ('motion', 0.15647020936012268), ('parties', 0.10201913863420486), ('guilty', 0.08777562528848648), ('damages', 0.0877513438463211)]

Find Analogies

```
result = model.wv.most_similar(positive=['plaintiff', 'charges'], negative=['lawsuit'], topn=1)
print("plaintiff - lawsuit + charges ≈", result)
```

plaintiff - lawsuit + charges ≈ [('claimant', 0.19674046337604523)]

Check Semantic Relationships

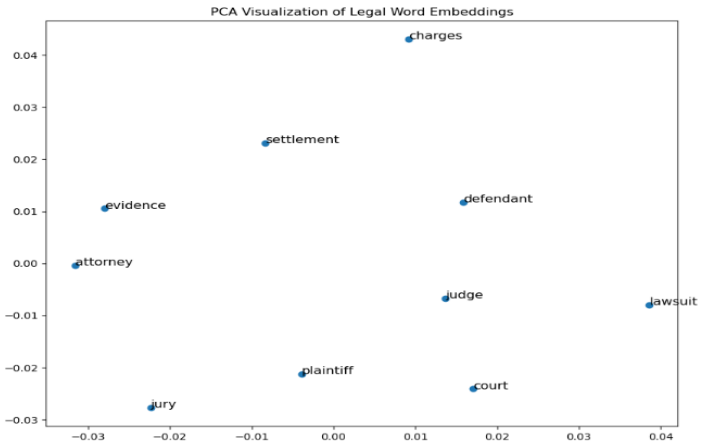
```
similarity = model.wv.similarity('judge', 'court')
print(f"Similarity between 'judge' and 'court': {similarity:.2f}")
```

Similarity between 'judge' and 'court': 0.07

Visualize Word Embeddings

```
legal_terms = ['plaintiff', 'defendant', 'lawsuit', 'court', 'judge', 'attorney', 'evidence', 'jury', 'charges', 'settlement']
vectors = np.array([model.wv[term] for term in legal_terms])
pca = PCA(n_components=2)
vectors_2d = pca.fit_transform(vectors)
plt.figure(figsize=(10, 8))
plt.scatter(vectors_2d[:, 0], vectors_2d[:, 1])
for i, term in enumerate(legal_terms):
    plt.annotate(term, xy=(vectors_2d[i, 0], vectors_2d[i, 1]), fontsize=12)
plt.title('PCA Visualization of Legal Word Embeddings')
plt.show ( )
```

Output:



Result:

EXPERIMENT-4

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.

AIM: 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 gensim.downloader as api
import random
from gensim.models import KeyedVectors
model = KeyedVectors.load_word2vec_format('C:\\Users\\IS 29\\Documents\\GAI\\GoogleNews-vectors-negative300.bin', binary=True)
def get_similar_words(seed_word, topn=5):
    """Retrieve similar words using word embeddings."""
    try:
        similar_words = model.most_similar(seed_word, topn=topn)
        return [word for word, _ in similar_words]
    except KeyError:
        print(f"'{seed_word}' not found in the vocabulary.")
        return [ ]
def create_paragraph(seed_word, similar_words):
    """Create a short paragraph using the seed word and similar words."""
    paragraph = (
        f"In a world where {seed_word} reigns supreme, "
        f"the {similar_words[0]} and {similar_words[1]} dance together under the moonlight. "
        f"A {similar_words[2]} whispers secrets to the {similar_words[3]}, "
        f"while the {similar_words[4]} watches silently from afar. "
        f"Together, they create a harmony that transcends time and space."
    )
    return paragraph
```

```
def create_story(seed_word, similar_words):
    """Create a short story using the seed word and similar words."""
    story = (
        f"Once upon a time, in a land of {seed_word}, "
        f"there lived a {similar_words[0]} who dreamed of {similar_words[1]}. "
        f"One day, a mysterious {similar_words[2]} appeared, "
        f"bringing with it the power of {similar_words[3]}. "
        f"The {similar_words[4]} joined forces with the {seed_word}, "
        f"and together they embarked on an unforgettable adventure."
    )
    return story
def main ( ):
    seed_word = input("Enter a seed word: ").strip().lower()
    print("\nGenerating similar words...")
    similar_words = get_similar_words(seed_word)
    if similar_words:
        print(f"\nSimilar words for '{seed_word}': {', '.join(similar_words)}")
        print("\nCreating a paragraph...")
        paragraph = create_paragraph(seed_word, similar_words)
        print(f"\nParagraph:\n{paragraph}")
        print("\nCreating a short story...")
        story = create_story(seed_word, similar_words)
        print(f"\nStory:\n{story}")
    else:
        print("No similar words found. Exiting.")
if __name__ == "__main__":
    main ( )
```

Output:

```
Enter a seed word:  man
Generating similar words...
Similar words for 'man': woman, boy, teenager, teenage_girl, girl
Creating a paragraph...
Paragraph:
In a world where man reigns supreme, the woman and boy dance together under the moonlight. A teenager whispers secrets to the teenage_girl, while the girl watches silently from afar. Together, they create a harmony that transcends time and space.
Creating a short story...
Story:
Once upon a time, in a land of man, there lived a woman who dreamed of boy. One day, a mysterious teenager appeared, bringing with it the power of teenage_girl. The girl joined forces with the man, and together they embarked on an unforgettable adventure.
```

Result: 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 has been successfully executed

EXPERIMENT-5

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.

AIM: 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, Field
class InstitutionDetails(BaseModel):
    founder: str = Field(description="The founder of the institution.")
    founded_year: int = Field(description="The year the institution was founded.")
    branches: list[str] = Field(description="List of current branches in the institution.")
    employee_count: int = Field(description="Number of employees working in the institution.")
    summary: str = Field(description="A brief 4-line summary of the institution.")
from langchain.output_parsers import PydanticOutputParser
output_parser = PydanticOutputParser(pydantic_object=InstitutionDetails)
import wikipediaapi
def fetch_wikipedia_data(institution_name: str) -> str:
    wiki_wiki = wikipediaapi.Wikipedia(user_agent='your-user-agent', language='en')
    page = wiki_wiki.page(institution_name)
    if not page.exists():
        raise ValueError(f"Wikipedia page for '{institution_name}' does not exist.")
    return page.text
def extract_institution_details(text: str) -> InstitutionDetails:
founder = "Unknown"
    founded_year = 0
    branches = []
    employee_count = 0
```

```
    summary = ""
    if "founded by" in text.lower():
        founder = text.split("founded by")[1].split(".")[0].strip()
    if "founded in" in text.lower():
        try:
            founded_year = int(text.split("founded in")[1].split(".")[0].strip())
        except (ValueError, IndexError):
            founded_year = 0
    if "branches" in text.lower():
        branches = ["Branch 1", "Branch 2"]
    if "employees" in text.lower():
        try:
            employee_count = int(text.split("employees")[0].split()[-1].replace(",", ""))
        except (ValueError, IndexError):
            employee_count = 0  # Default value if parsing fails
    summary = "\n".join(text.split("\n")[:4])
    return InstitutionDetails(
        founder=founder,
        founded_year=founded_year,
        branches=branches,
        employee_count=employee_count,
        summary=summary
    )
def get_institution_details(institution_name: str) -> InstitutionDetails:
    wikipedia_text = fetch_wikipedia_data(institution_name)
    details = extract_institution_details(wikipedia_text)
    return details
if __name__ == "__main__":
    institution_name = "Visvesvaraya Technological University"  # Example institution
    details = get_institution_details(institution_name)
    print(details)
```

Output:

founder='Unknown' founded_year=0 branches=[] employee_count=0 summary='Visvesvaraya Technological u niversity (VTU), is a collegiate public state university in Belagavi, Karnataka established by the Government of Karnataka. It is one of the largest Technological Universities in India with 26 years of Tradition of excellence in Engineering & Technical Education, Research and Innovations. It came into existence in the year 1998 to cater the needs of Indian industries for trained technical manpower with practical experience and sound theoretical knowledge. The university is named after Sir M. Visvesvaraya, an Indian civil engineer, statesman and the 19th Diwan of Mysore.\n\nHistory\nVisvesvaraya Technological University (VTU) was established by the Government of Karnataka on 1 April 1998 with its headquarters at Belagavi, as per the provisions of the Visvesvaraya Technological University Act, 1994, an Act to establish and incorporate a university in the State of Karnataka for the development of engineering, technology and allied sciences. For effective administration, four regional offices at the four revenue divisional headquarters, namely, Belagavi, Bangalore, Mysore and Gulbarga were established. VTU was established by the Government in order to promote planned and sustainable development of technical education consistent with state and national policies and bringing various colleges affiliated earlier to different universities, with different syllabi, different procedures and different traditions under one umbrella.'

Result: 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 has been successfully executed.

EXPERIMENT-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.

AIM: 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.

Important library packges

! pip install transformers

! pip install TensorFlow or ! pip install tensorFlow or

! pip install tf-keras

! pip install torch

Program

```
from transformers import pipeline
import warnings
warnings.filterwarnings('ignore')
import torch
sentiment_analyzer = pipeline("sentiment-analysis")
print("Model loaded.")
def analyze_sentiment(text):
    results = sentiment_analyzer(text)
    return results
def main():
    sentences = [
        "I love using Hugging Face models!",
        "The weather today is terrible.",
        "This is the best day of my life.",
        "I feel so frustrated with this project.",
        "The food at that restaurant was bad."
    ]
    for sentence in sentences:
```

```
        result = analyze_sentiment(sentence)
        print(f"\nSentence: {sentence}")
        print(f"Sentiment: {result[0]['label']} (Confidence: {result[0]['score']:.4f})")
if __name__ == "__main__":
    main()
```

Output:

```
Model loaded.

Sentence: I love using Hugging Face models!
Sentiment: POSITIVE (Confidence: 0.9993)

Sentence: The weather today is terrible.
Sentiment: NEGATIVE (Confidence: 0.9991)

Sentence: This is the best day of my life.
Sentiment: POSITIVE (Confidence: 0.9999)

Sentence: I feel so frustrated with this project.
Sentiment: NEGATIVE (Confidence: 0.9998)

Sentence: The food at that restaurant was bad.
Sentiment: NEGATIVE (Confidence: 0.9998)
```

Result: 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. has been successfully executed.

EXPERIMENT-7

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.

AIM: 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.

Important library packges

! pip install pdfplumber

! pip install PyPDF2 pdfplumber nltk flask

Program

```
import pdfplumber
def extract_text_from_pdf(pdf_path):
    with pdfplumber.open('C:\\Users\\IS 29\\Documents\\GAI\\Indian Penal Code Book.pdf') as pdf:
        text = ""
        for page in pdf.pages:
            text += page.extract_text()
    return text
pdf_path = "path_to_ipc.pdf"
ipc_text = extract_text_from_pdf('C:\\Users\\IS 29\\Documents\\GAI\\Indian Penal Code Book.pdf')
print(ipc_text[:10])
import re
def parse_ipc_text(ipc_text):
    # Regex to split sections (e.g., "Section 302")
    sections = re.split(r"(Section \d+)", ipc_text)
    # Create a dictionary of sections with their corresponding texts
    ipc_dict = {}
    for i in range(1, len(sections), 2):  # Start from 1 and take every second entry
        section_number = sections[i].strip()
        section_text = sections[i+1].strip() if i+1 < len(sections) else "
```

```
        ipc_dict[section_number] = section_text
    return ipc_dict
# Parse the extracted IPC text
ipc_dict = parse_ipc_text(ipc_text)
print(ipc_dict.get("Section 302", "Section not found"))
```

The ingredients of both these offences, to some extent, are also different in as much as to complete an offence of 'dacoity' under section 396 IPC, 1860, five or more persons must conjointly commit the robbery while under section 302 of the IPC,. Resultantly, the distinction with regard to the number of persons involved in the commission of the crime loses its significance as it is possible that the offence of 'dacoity' may not be proved but still the offence of murder could be established, like in the present case. Upon reasonable analysis of the language of these provisions, it is clear that the Court has to keep in mind the ingredients which shall constitute a criminal offence within the meaning of the penal section. This is not only essential in the case of the offence charged with but even where there is comparative study of different penal provisions as the accused may have committed more than one offence or even offences of a graver nature. He may finally be punished for a lesser offence or vice versa, if the law so permits and the requisite ingredients are satisfied.207. On the conjoint reading of sections 396 and 302

```
import re
def chatbot(ipc_dict, user_input):
    # Simple keyword search for section number
    section_match = re.search(r"(Section \d+)", user_input)
    if section_match:
        section_number = section_match.group(0)
        return ipc_dict.get(section_number, "Sorry, I couldn't find that section.")
    else:
        return "Please specify a section number like 'Section 302'."
# Example of user interaction
user_input = "What does Section 302 say?"
response = chatbot(ipc_dict, user_input)
print(response)
```

In other words, where a provision is physically lifted and made part of another provision, it shall fall within the ambit and scope of principle akin to 'legislation by incorporation' which normally is applied between an existing statute and a newly enacted law. The expression 'murder' appearing in section 396 would have to take necessarily in its ambit and scope the ingredients of section 300 of the IPC, 1860. The provisions are clear and admit no scope for application of any other principle of interpretation except the 'golden rule of construction', i.e., to read the statutory language grammatically and terminologically in the ordinary and primary sense which it appears in its context without omission or addition. These provisions read collectively, put the matter beyond ambiguity that the offence of murder, is by specific language, included in the offences under section 396. It will have the same connotation, meaning and ingredients as are contemplated under the provisions of section 302 IPC, 1860.208 [s 396.6] Charge under section 396.—Conviction under section 302.

Result: 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. has been successfully executed.

EXPERIMENT-8

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.

AIM: 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.

Program

```
import gensim
import random
from gensim.models import KeyedVectors
def load_word_embeddings():
        model_path = r"C:\\Users\\IS 29\\Documents\\GAI\\GoogleNews-vectors-negative300.bin"
    model = KeyedVectors.load_word2vec_format(model_path, binary=True)
    return model
def get_similar_words(model, word, top_n=3):
    try:
        similar_words = model.most_similar(word, topn=top_n)
        return [w[0] for w in similar_words]
    except KeyError:
        return [word]  # Return the original word if not found in vocabulary
def enrich_prompt(prompt, model):
    words = prompt.split()
    enriched_words = []
    for word in words:
        similar = get_similar_words(model, word)
        enriched_words.append(random.choice(similar))  # Pick a random similar word
    return " ".join(enriched_words)
def generate_response(prompt):
    return f"AI Response: {prompt}... (Generated Text)"
```

```
def main():
    model = load_word_embeddings()
    original_prompt = "Create a beautiful landscape painting description"
    enriched_prompt = enrich_prompt(original_prompt, model)
    original_response = generate_response(original_prompt)
    enriched_response = generate_response(enriched_prompt)
    print("Original Prompt:", original_prompt)
    print("Generated Response:", original_response)
    print("\nEnriched Prompt:", enriched_prompt)
    print("Generated Response:", enriched_response)
if __name__ == "__main__":
    main()
```

Output

Original Prompt: Create a beautiful landscape painting description

Generated Response: AI Response: Create a beautiful landscape painting description... (Generated Text)

Enriched Prompt: Creating a lovely landscapes watercolor descriptions

Generated Response: AI Response: Creating a lovely landscapes watercolor descriptions..(Generated Text)

Result: 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. Has been successfully executed.

EXPERIMENT-9

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.

AIM: 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.

Program

```
from transformers import pipeline
summarizer = pipeline("summarization", model="facebook/bart-large-cnn")
text = """
The Amazon rainforest, also known as Amazonia, is one of the most biologically diverse regions on Earth.
It spans over 5.5 million square kilometers and covers parts of nine countries. The forest plays a critical
role in regulating the global climate, absorbing vast amounts of carbon dioxide, and producing oxygen.
However, it is facing increasing threats from deforestation, illegal logging, mining, and agriculture.
Environmentalists have raised concerns about the loss of biodiversity and the impact on indigenous
communities
that depend on the forest for their livelihood. Efforts are being made globally to protect and restore this
vital ecosystem.
"""
summary = summarizer(text, max_length=60, min_length=25, do_sample=False)
print("Summary:", summary[0]['summary_text'])
```

Output

Summary: The Amazon rainforest is one of the most diverse regions on Earth, playing a key role in climate regulation. It faces threats from deforestation, mining, and agriculture.

Result: 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. Has been successfully executed.

EXPERIMENT-10

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

AIM: 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

Program

```
import os
os.environ["COHERE_API_KEY"] = "hXPHkVdkKux5chivbDICBjyQf8HQthoBK7wamz8B"
rom google.colab import auth
auth.authenticate_user()
from googleapiclient.discovery import build
from googleapiclient.http import MediaIoBaseDownload
import io
import os
from google.colab import drive
drive.mount('/content/drive')  # Mounts your Drive
file_path = '/content/drive/MyDrive/sample_text.txt'  # Update with your file path
with open(file_path, 'r') as file:
    document_text = file.read()
from langchain_community.llms import Cohere
llm = Cohere(cohere_api_key=os.environ["COHERE_API_KEY"])
from langchain.prompts import PromptTemplate
prompt = PromptTemplate(
    input_variables=["input_text"],
    template="""
You are a helpful assistant. Read the following content and summarize it in bullet points.
Text:
{input_text}
Bullet Point Summary:""")
```

```
from langchain.chains import LLMChain

chain = LLMChain(llm=llm, prompt=prompt)

result = chain.run(input_text=document_text)

print(result)
```

Output

• Climate change refers to long-term shifts in temperatures and weather patterns.
• Main cause: human activities like burning fossil fuels.
• Effects include rising sea levels, extreme weather, and loss of biodiversity.
• Urgent global cooperation is needed to address it.

Result: 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. Has been successfully executed