Program 1

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

In [ ]:
```
!pip install gensim
```

In [ ]:
```
import gensim.downloader as api

model = api.load("glove-wiki-gigaword-50")

word1 = "king"
word2 = "man"
word3 = "woman"

result_vector = model[word1]- model[word2] + model[word3]
predicted_word = model.most_similar([result_vector], topn=5)
```

In [ ]:
```
print("Top 5 similar words\n")
for word, similarity in predicted_word:
    print(f"{word}:{similarity:.2f}")
```

In [ ]:
```
print(f"Result of '{word1}- {word2} + {word3}' is: {predicted_word[1][0]}")
```

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.

In [ ]:
```
!pip install gensim
```

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

model = api.load("glove-wiki-gigaword-50")

words = ["computer", "internet", "software", "hardware", "disk", "robot", "data"
         "network", "cloud", "algorithm"]

word_vectors = np.array([model[word] for word in words])

pca = PCA(n_components=2)
reduced_vectors = pca.fit_transform(word_vectors)
```

In [ ]:
```
plt.figure(figsize=(8, 6))
for i, word in enumerate(words):
    plt.scatter(reduced_vectors[i, 0], reduced_vectors[i, 1])
    plt.annotate(word, (reduced_vectors[i, 0], reduced_vectors[i, 1]))

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

In [ ]:
```
input_word = "computer"
similar_words = model.most_similar(input_word, topn=5)
print("Top 5 similar words are:\n")
for word, similarity in similar_words:
    print(f"{word}:{similarity:.2f}")
```

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.

In [ ]:
```
!pip install gensim nltk
```

In [ ]:
```
from gensim.models import Word2Vec
import nltk
from nltk.tokenize import word_tokenize

nltk.download('punkt')

corpus = [
    "A patient with diabetes requires regular insulin injections.",
    "Medical professionals recommend exercise for heart health.",
    "Doctors use MRI scans to diagnose brain disorders.",
    "Antibiotics help fight bacterial infections but not viral infections.",
    "The surgeon performed a complex cardiac surgery successfully.",
    "Doctors and nurses work together to treat patients.",
    "A doctor specializes in diagnosing and treating diseases."
]

tokenized_corpus = [word_tokenize(sentence.lower()) for sentence in corpus]

model = Word2Vec(
    sentences=tokenized_corpus,
    vector_size=100,
    window=3,
    min_count=1,
    workers=4,
    sg=1
)
```

In [ ]:
```
similar_words = model.wv.most_similar("doctor", topn=5)

print("Top 5 words similar to 'doctor':")
for word, score in similar_words:
    print(f"{word}: {score:.4f}")
```

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.

In [ ]:
```
!pip install gensim cohere
```

In [ ]:
```
import os
import gensim.downloader as api
import cohere

co = cohere.Client("3JE437qflNp5M7TbvoQXMmeDuRrRWgylGCq09x29")
model = api.load("glove-wiki-gigaword-50")
```

In [ ]:
```
input_word = "technology"
similar_words = model.most_similar(input_word, topn=5)
print("Top 5 similar words are:\n")
for word, similarity in similar_words:
    print(f"{word}:{similarity:.2f}")
```

In [ ]:
```
related_terms = [input_word] + [w[0] for w in similar_words]
enriched_prompt = (
    "Explain the impact of " + ", ".join(related_terms[:-1]) +
    f", and {related_terms[-1]} on society."
)
```

In [ ]:
```
def generate_response(prompt):
    response = co.generate(
        model="command",
        prompt=prompt,
        max_tokens=150
    )
    return response.generations[0].text.strip()
```

In [ ]:
```
original_prompt = "Explain the impact of technology on society."
original_response = generate_response(original_prompt)
enriched_response = generate_response(enriched_prompt)
```

In [ ]:
```
print("Original Prompt Response:")
print(original_response)
```

In [ ]:
```
print("Enriched Prompt Response:")
print(enriched_response)
```

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.

In [ ]:
```
!pip install gensim nltk
```

In [ ]:
```
import gensim.downloader as api
model = api.load("glove-wiki-gigaword-50")
```

In [ ]:
```
def construct_paragraph(seed_word, similar_words):
    paragraph = f"""
        In the spirit of {seed_word}, one might embark on an unforgettable
        {similar_words[0][0]} to distant lands. Every {similar_words[1][0]}
        brings new challenges and opportunities for {similar_words[2][0]}.
        Through perseverance and courage, the {similar_words[3][0]} becomes
        a tale of triumph, much like an {similar_words[4][0]}.
    """
    return paragraph
```

In [ ]:
```
input_word = "adventure"
similar_words = model.most_similar(input_word, topn=5)
```

In [ ]:
```
paragraph = construct_paragraph(input_word, similar_words)
print(paragraph)
```

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.

In [ ]:
```
!pip install transformers torch
```

In [ ]:
```
from transformers import pipeline

sentiment_pipeline = pipeline("sentiment-analysis")
sentences = [
    "I love this product! It works perfectly.",
    "This is the worst experience I've ever had.",
    "The weather is nice today.",
    "I feel so frustrated with this service."
]
```

In [ ]:
```
results = sentiment_pipeline(sentences)
print("Sentiment Analysis Results:\n")
for sentence, result in zip(sentences, results):
    print(f"Sentence: {sentence}")
    print(f"Sentiment: {result['label']}, Confidence: {result['score']:.4f}")
    print()
```

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.

In [ ]:
```
!pip install transformers torch
```

In [ ]:
```
from transformers import pipeline

summarizer = pipeline("summarization", model="t5-small", tokenizer="t5-small")
passage = """
    Machine learning is a subset of artificial intelligence that focuses on
    training algorithms to make predictions or decisions without being
    explicitly programmed. It is widely used in industries such as healthcare,
    where it helps in early diagnosis, in finance for fraud detection, and in
    retail for customer behavior prediction. As data continues to grow, machine
    learning models are becoming increasingly important for extracting
    meaningful patterns and making data-driven decisions.
"""
```

In [ ]:
```
summary = summarizer(passage, max_length=30, min_length=10, do_sample=False)

print("Summary:")
print(summary[0]['summary_text'])
```

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.

In [ ]:
```
!pip install langchain langchain-community cohere google-auth
```

In [ ]:
```
!pip google-auth oauthlib google-auth httplib2 googleapiclient
```

In [ ]:
```
import os
from cohere import Client
from langchain.prompts import PromptTemplate

co = Client("3JE437qflNp5M7TbvoQXMmeDuRrRWgylGCq09x29")
```

In [ ]:
```
text_document = """
Machine learning is a subset of artificial intelligence that focuses on training
algorithms to make predictions. It is widely used in industries like healthcare,
finance, and retail.
"""

template = """
You are an expert summarizer. Summarize the following text in a concise manner:
Text: {text}
Summary:
"""
```

In [ ]:
```
prompt_template = PromptTemplate(input_variables=["text"], template=template)
formatted_prompt = prompt_template.format(text=text_document)

response = co.generate(
    model="command",
    prompt=formatted_prompt,
    max_tokens=50
)
```

In [ ]:
```
print("Summary:")
print(response.generations[0].text.strip())
```

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.

In [ ]:
```
!pip install wikipedia cohere pydantic
```

In [ ]:
```
from typing import List
from pydantic import BaseModel
import wikipedia
import cohere
import json

co = cohere.Client("3JE437qflNp5M7TbvoQXMmeDuRrRWgylGCq09x29")
```

In [ ]:
```
class InstitutionDetails(BaseModel):
    institution_name: str
    founded_year: int
    founded_by: str
    no_of_employees: str
    branches: List[str]
    courses: List[str]

details_format = {
    "institution_name": "str",
    "founded_year" :"int",
    "founded_by": "str",
    "no_of_employees": "str",
    "branches": "List[str]",
    "courses": "List[str]",
}
```

In [ ]:
```
def fetch_wikipedia_summary(name: str) -> str:
    return wikipedia.summary(name, sentences=4)
```

In [ ]:
```
def format_wiki_summary(summary):
    response = co.generate(prompt=f"""
    Summarize the following text: ||{summary}||
    in this json format {details_format}
    """)
    text = response.generations[0].text.strip()

    start = text.find("{")
    end = text.rfind("}") + 1
    json_str = text[start:end].replace("'",'"')

    data = json.loads(json_str)
    return InstitutionDetails(**data)
```

In [ ]:
```
def get_institution_details() -> InstitutionDetails:
    name = input("Enter institution name: ")
    summary = fetch_wikipedia_summary(name)
    return format_wiki_summary(summary)
```

In [ ]:
```
institution = get_institution_details()
print("Institution Name: ", institution.institution_name)
print("Founded in: ", institution.founded_year)
print("Founded by: ", institution.founded_by)
print("Employee count: ", institution.no_of_employees)
print("Departments: ", institution.branches)
print("Courses Offered: ", institution.courses)
```

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.

In [ ]:
```
!pip install cohere pypdf ipywidgets
```

In [ ]:
```
import cohere
from pypdf import PdfReader
import getpass

co = cohere.Client("3JE437qflNp5M7TbvoQXMmeDuRrRWgylGCq09x29")
reader = PdfReader("./IPC_186045.pdf")          # copy paste path of the file
```

In [ ]:
```
text =""
for page in reader.pages:
    text += page.extract_text()
```

In [ ]:
```
def ask_ipc(question):
    prompt = f"""
    You are a legal assistant specialized in the Indian Penal Code (IPC).
    Use the following content to answer the user's question:

    {text[:10000]}

    User Question: {question}

    Respond with a relevant IPC section (if any) and a clear explanation.
    """
    response = co.generate(
        prompt=prompt,
        model="command-r-plus",
        max_tokens=300
    )
    print("\n" + response.generations[0].text.strip())
```

In [ ]:
```
user_input = input("Ask your IPC question: ")
ask_ipc(user_input)
```