**#1**

from gensim.downloader import load

print("Loading pre-trained GloVe model (50 dimensions)...")

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

def ewr():

result = model.most\_similar(positive=["king", "woman"], negative=["man"], topn=1)

print("\n'king' - 'man' + 'woman'?", result[0][0])

print("Similarity:", result[0][1])

result = model.most\_similar(positive=["paris", "italy"], negative=["france"], topn=1)

print("\n'paris' - 'france' + 'italy'?", result[0][0])

print("Similarity:", result[0][1])

result = model.most\_similar(positive=["programming"], topn=5)

print("\nTop 5 words similar to 'programming':")

for word, similarity in result:

print(word, similarity)

ewr()

**#2**

import gensim.downloader as api

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

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

tech\_words = ['computer', 'internet', 'software', 'hardware', 'keyboard', 'network', 'algorithm', 'data', 'program', 'database']

vectors = [wv[word] for word in tech\_words]

pca = PCA(n\_components=2)

points = pca.fit\_transform(vectors)

plt.figure(figsize=(8, 6))

for i, word in enumerate(tech\_words):

x, y = points[i]

plt.scatter(x, y)

plt.text(x + 0.01, y + 0.01, word)

plt.title("Technology Word Embeddings (PCA)")

plt.xlabel("PCA 1")

plt.ylabel("PCA 2")

plt.grid(True)

plt.show()

def similar\_words(word):

try:

print(f"\nTop 5 words similar to '{word}':")

for w, sim in wv.most\_similar(word, topn=5):

print(f"{w}: {sim:.4f}")

except KeyError:

print(f"'{word}' not found in vocabulary.")

similar\_words("network")

**#3**

import gensim

from gensim.models import Word2Vec

import matplotlib.pyplot as plt

from sklearn.manifold import TSNE

import numpy as np

# Sample medical corpus

corpus = [

"The patient was diagnosed with diabetes and hypertension.",

"MRI scans reveal abnormalities in the brain tissue.",

"The treatment involves antibiotics and regular monitoring.",

"Symptoms include fever, fatigue, and muscle pain.",

"The vaccine is effective against several viral infections.",

"Doctors recommend physical therapy for recovery.",

"Clinical trial results were published in the journal.",

"The surgeon performed a minimally invasive procedure.",

"Prescription includes pain relievers and anti-inflammatory drugs.",

"Diagnosis confirmed a rare genetic disorder."

]

sentences = [sentence.lower().split() for sentence in corpus]

model = Word2Vec(sentences, vector\_size=100, window=5, min\_count=1, epochs=50)

words = list(model.wv.index\_to\_key)

word\_vectors = np.array([model.wv[word] for word in words])

tsne = TSNE(n\_components=2, perplexity=5, random\_state=42, n\_iter=300)

reduced\_vectors = tsne.fit\_transform(word\_vectors)

plt.figure(figsize=(10, 8))

plt.scatter(reduced\_vectors[:, 0], reduced\_vectors[:, 1])

for i, word in enumerate(words):

plt.text(reduced\_vectors[i, 0] + 0.02, reduced\_vectors[i, 1] + 0.02, word, fontsize=9)

plt.title("t-SNE of Medical Word Embeddings")

plt.grid(True)

plt.show()

def find\_similar(word):

try:

similar\_words = model.wv.most\_similar(word, topn=5)

print(f"\nWords similar to '{word}':")

for similar, score in similar\_words:

print(f"{similar}: {score:.2f}")

except KeyError:

print(f"'{word}' not found in vocabulary.")

find\_similar("treatment")

find\_similar("vaccine")

**#4**

!pip install gensim transformers nltk torch

import string

import nltk

from nltk.tokenize import word\_tokenize

from gensim.downloader import load

from transformers import pipeline

nltk.download('punkt')

print("Loading resources...")

wv = load("glove-wiki-gigaword-100")

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

def enrich\_prompt(prompt, keyword):

words = word\_tokenize(prompt)

new\_words = []

for word in words:

clean = word.lower().strip(string.punctuation)

if clean == keyword.lower():

try:

similar = wv.most\_similar(clean, topn=1)[0][0]

print(f"Replacing '{word}' → '{similar}'")

new\_words.append(similar)

except KeyError:

new\_words.append(word)

else:

new\_words.append(word)

return ' '.join(new\_words)

def generate(prompt):

return generator(prompt, max\_length=100)[0]['generated\_text']

original = "Who is king."

print(f"\nOriginal Prompt: {original}")

enriched = enrich\_prompt(original, "king")

print(f"Enriched Prompt: {enriched}")

print("\nGenerating GPT-2 responses...")

r1 = generate(original)

r2 = generate(enriched)

print("\nOriginal Prompt Response:\n", r1)

print("\nEnriched Prompt Response:\n", r2)

print("\n--- Comparison ---")

print("Original Length:", len(r1))

print("Enriched Length:", len(r2))

print("Original Sentence Count:", r1.count('.'))

print("Enriched Sentence Count:", r2.count('.'))

**#5**

import gensim.downloader as api

import nltk, random

nltk.download('punkt')

wv = api.load("glove-wiki-gigaword-100")

def similar\_words(word, topn=5):

try:

return [w for w, \_ in wv.most\_similar(word, topn=topn)]

except KeyError:

return []

def make\_sentence(word, sim):

templates = [

f"The {word} is associated with {sim[0]} and {sim[1]}.",

f"People often link '{word}' to {sim[2]}.",

f"{word} and {sim[3]} often go together.",

f"In {word} studies, {sim[4]} is vital."

]

return random.choice(templates)

def make\_paragraph(word):

sim = similar\_words(word)

if not sim: return "Try another seed word."

return ' '.join(make\_sentence(word, sim) for \_ in range(4))

print("\nGenerated Paragraph:\n")

print(make\_paragraph("river"))

**#6**

from transformers import pipeline

analyzer = pipeline("sentiment-analysis")

def analyze(text):

result = analyzer(text)[0]

print(f"\nText: {text}")

print(f"Sentiment: {result['label']} (Confidence: {result['score']:.4f})")

reviews = [

"This app is amazing! I love it so much.",

"I'm very disappointed, the service was terrible.",

"Absolutely fantastic, nothing special.",

"Not useful or helpful. Highly not recommended.",

"It great, but not the worst either."

]

print("Customer Feedback Analysis:")

for r in reviews:

analyze(r)

**#7**

from transformers import pipeline

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

while True:

text = input("\nPaste passage (or type 'exit' to quit):\n").strip()

if text.lower() == 'exit':

print("Exiting summarization.")

break

if len(text.split()) < 50:

print("Please enter at least 50 words.")

continue

summary = summarizer(text, max\_length=130, min\_length=30, do\_sample=False)[0]['summary\_text']

print("\n Summary:\n" + summary)

**#8**

!pip install langchain cohere langchain-community

import cohere

import getpass

from langchain.prompts import PromptTemplate

from langchain\_community.llms import Cohere

file\_path = "teaching.txt"

try:

with open(file\_path, "r", encoding="utf-8") as file:

text\_content = file.read()

print("File loaded successfully!")

except Exception as e:

print("Error loading file:", str(e))

text\_content = ""

COHERE\_API\_KEY = getpass.getpass("Enter your Cohere API Key: ")

cohere\_llm = Cohere(cohere\_api\_key=COHERE\_API\_KEY, model="command")

template = """

You are an AI assistant helping to summarize and analyze a text document.

Here is the document content:

{text}

Summary:

- Provide a concise summary of the document.

Key Takeaways:

- List 3 important points from the text.

Sentiment Analysis:

- Determine if the sentiment of the document is Positive, Negative, or Neutral.

"""

prompt\_template = PromptTemplate(

input\_variables=["text"],

template=template

)

formatted\_prompt = prompt\_template.format(text=text\_content)

response = cohere\_llm.predict(formatted\_prompt)

print("\n\*Formatted Output\*\n")

print(response)

**API KEY: 09zel9GU21Ct90waRKFKbBKqkH9HjOq4W4aKRQWx**

**Inside text file:** Artificial Intelligence (AI) is a rapidly evolving field that focuses on building intelligent systems capable of performing tasks that typically require human intelligence. These tasks include reasoning, learning, problem-solving, perception, and language understanding. AI has various subfields such as machine learning, deep learning, and natural language processing. It has significant applications in domains like healthcare, finance, education, and autonomous systems. Despite its many advantages, AI raises concerns about ethics, privacy, and employment. It is important to ensure responsible development of AI by incorporating transparency, fairness, and human-centered design principles.

**#9**

!pip install wikipedia-api

import wikipediaapi

from pydantic import BaseModel

class InstitutionDetails(BaseModel):

name: str

founder: str = "Not Available"

founded\_year: str = "Not Available"

branches: str = "Not Available"

employees: str = "Not Available"

summary: str

def fetch\_institution\_details(name: str) -> InstitutionDetails:

wiki = wikipediaapi.Wikipedia(user\_agent="MyWikiBot/1.0", language="en")

page = wiki.page(name)

if not page.exists():

raise ValueError("Institution page does not exist on Wikipedia")

summary = ". ".join(page.summary.split(".")[:4]) + "."

details = {"name": name, "summary": summary}

for section in page.sections:

key = section.title.lower()

if "founder" in key:

details["founder"] = section.text.split(".")[0]

elif "founded" in key:

details["founded\_year"] = section.text.split(".")[0]

elif "branches" in key:

details["branches"] = section.text.split(".")[0]

elif "employees" in key:

details["employees"] = section.text.split(".")[0]

return InstitutionDetails(\*\*details)

name = input("Enter Institution Name: ")

try:

inst = fetch\_institution\_details(name)

print(inst.model\_dump\_json(indent=4))

except ValueError as e:

print(e)

**#10**

!pip install PyPDF2 nltk scikit-learn

import PyPDF2, re, string

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

nltk.download('punkt')

nltk.download('stopwords')

def extract\_text(pdf\_path):

try:

with open(pdf\_path, 'rb') as f:

reader = PyPDF2.PdfReader(f)

return "".join(page.extract\_text() or "" for page in reader.pages)

except:

return ""

def create\_index(text):

pattern = r"((?:CHAPTER|SECTION)\s+\w+\.?\s+.\*?)(?=(?:CHAPTER|SECTION)\s+\w+\.?\s+|$)"

index = {}

for match in re.findall(pattern, text, re.DOTALL | re.IGNORECASE):

title = re.match(r"(?:CHAPTER|SECTION)\s+\w+\.?\s+(.\*?)(?=\n)", match, re.I)

if title:

index[title.group(1).strip()] = match[title.end():].strip()

return index

def find\_section(query, index):

sections = list(index.values())

titles = list(index.keys())

vec = TfidfVectorizer(stop\_words='english')

tfidf = vec.fit\_transform(sections + [query])

sim = cosine\_similarity(tfidf[-1], tfidf[:-1]).flatten()

if sim.max() == 0:

return None

return titles[sim.argmax()]

def chatbot(index):

print("IPC Chatbot started. Type 'exit' to quit.")

while True:

q = input("You: ")

if q.lower() == 'exit': break

sec = find\_section(q, index)

if sec:

print(f"Chatbot: Section \*\*{sec}\*\*\n{index[sec][:1000]}...\n")

else:

print("Chatbot: No relevant info found.\n")

if \_\_name\_\_ == "\_\_main\_\_":

text = extract\_text("ipc.pdf")

if text:

idx = create\_index(text)

chatbot(idx)

else:

print("Failed to load IPC PDF.")