|  |  |
| --- | --- |
| 1)  import os  import gensim.downloader as api  import numpy as np  import matplotlib.pyplot as plt  from sklearn.decomposition import PCA  from gensim.models import KeyedVectors  from google.colab import drive drive.mount('/content/drive') model\_path = "/content/drive/My Drive/word2vec-google-news-300.model" if os.path.exists(model\_path):  print("Model found in Google Drive..Loading")  word\_vectors = KeyedVectors.load(model\_path)  else:  print("Model not found. Downloading Word2Vec model...")  word\_vectors = api.load("word2vec-google-news-300")  print("Saving model to Google Drive for future use...")  word\_vectors.save(model\_path)  print("Model saved successfully")  print("\nModel Loaded Successfully\n") print("Top 5 words similar to 'computer':")  similar\_words = word\_vectors.most\_similar("computer", topn=5)  for word, similarity in similar\_words:  print(f"{word}: {similarity:.4f}") print("\nPerforming Vector Arithmetic: 'king - man + woman'")  # king - man + woman = ?  result = word\_vectors.most\_similar(positive=['king', 'woman'], negative=['man'], topn=1)  print(f"Result: {result[0][0]}") # Expected output: 'queen' print("\n More Examples of Vector Arithmetic:")  examples = [  ("Paris", "France", "Italy"),  ("Einstein", "scientist", "painter")  ]  for w1, w2, w3 in examples:  result = word\_vectors.most\_similar(positive=[w1, w3], negative=[w2], topn=1)  print(f"{w1} - {w2} + {w3} = {result[0][0]}") | **2)**  **import gensim**  **import gensim.downloader as api**  **import numpy as np**  **import matplotlib.pyplot as plt**  **from sklearn.decomposition import PCA**  **from sklearn.manifold import TSNE**  **import os**  ***# Define model path in Google Drive***  model\_path = "/content/drive/My Drive/word2vec-google-news-300.model"  ***# Step 1: Mount Google Drive***  from google.colab import drive  drive.mount('/content/drive')  ***# Step 2: Check if model already exists***  if os.path.exists(model\_path):      print(" Model found! Loading the saved model...")      word2vec\_model = gensim.models.KeyedVectors.load(model\_path, mmap='r')  else:      print("Model not found. Downloading now...")  word2vec\_model = api.load("word2vec-google-news-300")  ***# Save the downloaded model to Google Drive***      print("Saving model to Google Drive...")      word2vec\_model.save(model\_path)      print(" Model saved successfully!")  **def get\_word\_vectors(model, words):**     return np.array([model[word] for word in words if word in model])  **def reduce\_dimensions(vectors, method='pca'):**      if method == 'pca':          reducer = PCA(n\_components=2)      elif method == 'tsne':          reducer = TSNE(n\_components=2, random\_state=42, perplexity=5)      else:          raise ValueError("Method should be 'pca' or 'tsne'")      return reducer.fit\_transform(vectors)  **def plot\_embeddings(words, reduced\_vectors, title):**      plt.figure(figsize=(10, 6))      for word, coord in zip(words, reduced\_vectors):          plt.scatter(coord[0], coord[1], marker='o')   plt.text(coord[0] + 0.01, coord[1] + 0.01, word, fontsize=12)      plt.title(title)      plt.xlabel("Dimension 1")      plt.ylabel("Dimension 2")      plt.grid()      plt.show()  **def find\_similar\_words(model, word, top\_n=5):**        if word in model:          similar\_words = model.most\_similar(word, topn=top\_n)          return [w[0] for w in similar\_words]      else:          return ["Word not in vocabulary"]  ***# Define 10 words from the technology domain***  tech\_words = ["computer", "software", "hardware", "algorithm", "internet",                "network", "data", "cloud", "AI", "machine"]  ***# Get word embeddings***  print("Fetching word embeddings...")  word\_vectors = get\_word\_vectors(word2vec\_model, tech\_words)  ***# Reduce dimensions using PCA***  print("Applying PCA...")  reduced\_vectors\_pca = reduce\_dimensions(word\_vectors, method='pca')  plot\_embeddings(tech\_words, reduced\_vectors\_pca, title="PCA Visualization of Word Embeddings")  ***# Reduce dimensions using t-SNE***  print("Applying t-SNE...")  reduced\_vectors\_tsne = reduce\_dimensions(word\_vectors, method='tsne')  plot\_embeddings(tech\_words, reduced\_vectors\_tsne, title="t-SNE Visualization of Word Embeddings")  ***# Find similar words***  input\_word = "computer"  print(f"Finding words similar to '{input\_word}'...")  similar = find\_similar\_words(word2vec\_model, input\_word)  print("Top similar words:", similar) |

|  |  |
| --- | --- |
| # Experiment 3: Word2Vec Training on Medical Corpus with Bigram Detection  # Import libraries  from gensim.models import Word2Vec  from gensim.models.phrases import Phrases, Phraser  import matplotlib.pyplot as plt  from sklearn.decomposition import PCA  # Read medical corpus from an external file  with open("medical\_corpus.txt", "r") as file:  corpus = [line.strip() for line in file if line.strip()]  # Refined stopword list stopwords = {"the", "a", "is", "for", "with", "to", "of", "and", "in", "can", "are"}  # Tokenization with stopword removal  tokenized\_sentences = [  [word for word in sentence.lower().split() if word not in stopwords]  for sentence in corpus  ]  # Detect bigrams to capture word combinations (e.g., "blood sugar", "high pressure")  bigram = Phrases(tokenized\_sentences, min\_count=2, threshold=5)  bigram\_phraser = Phraser(bigram)  tokenized\_sentences = [bigram\_phraser[sentence] for sentence in tokenized\_sentences]  # Train Word2Vec model with improved parameters  model = Word2Vec(tokenized\_sentences, vector\_size=150, window=5, min\_count=2, epochs=300, sg=1, hs=1,  negative=0)  # Display similar words  diabetes\_similar = [(word, round(sim, 2)) for word, sim in model.wv.most\_similar("diabetes", topn=5)]  print("Words similar to 'diabetes':", diabetes\_similar)  hypertension\_similar = [(word, round(sim, 2)) for word, sim in model.wv.most\_similar("hypertension",  topn=5)]  print("Words similar to 'hypertension':", hypertension\_similar) | 4)  %pip install numpy  %pip install scipy  %pip install gensim  import os  import gensim.downloader as api  fromgensim.modelsimport KeyedVectors  from google.colab import drive  drive.mount('/content/drive')  model\_path = "/content/drive/My Drive/word2vec-google-news-300.model"  if os.path.exists(model\_path):    print("Model found in Google Drive..Loading")    word\_vectors = KeyedVectors.load(model\_path)    print("Loading Completed")  else:    print("Model not found. Downloading Word2Vec model...")    word\_vectors=api.load("word2vec-google-news-300")    print("Saving model to Google Drive for future use...")    word\_vectors.save(model\_path)    print("Model saved successfully")    print("\nModel Loaded Successfully\n")  print(word\_vectors.most\_similar("king"))  original\_prompt = input("Enter the original prompt: ")  ***# Get key terms from user (comma-separated)***  key\_terms\_input = input("Enter key terms (comma-separated): ")  key\_terms = [term.strip() for term in key\_terms\_input.split(",")]  similar\_terms = []  for term in key\_terms:      if term in  word\_vectors.key\_to\_index:          similar\_terms.extend({word for word, \_ in word\_vectors.most\_similar(term, topn=2)})  if similar\_terms:      enriched\_prompt = f"{original\_prompt} Consideraspectslike:{','.join(similar\_terms)}."  else:      enriched\_prompt = original\_prompt  print("Original Prompt:", original\_prompt)  print("Enriched Prompt:", enriched\_prompt)  import getpass  import os  GOOGLE\_API\_KEY= os.environ["GOOGLE\_API\_KEY"] = getpass.getpass("Enter your Google AI API key: ")  %pip install langchain-google-genai  %pip install langchain-core  %pip install langchain-community  %pip install -qU langchain-google-genai  %pip install --upgrade langchain  from langchain\_google\_genai import ChatGoogleGenerativeAI  llm = ChatGoogleGenerativeAI(      model="gemini-2.0-flash-exp",      temperature=0.3,      api\_key=GOOGLE\_API\_KEY,      max\_tokens=512,      timeout=30,      max\_retries=2,  )  llm.invoke("Hi") print(llm.invoke(original\_prompt).content) print(llm.invoke(enriched\_prompt).content) |

|  |  |
| --- | --- |
| 5)  !pip install -q langchain-google-genai gradio  # Imports and Gemini LLM setup  from langchain\_google\_genai import ChatGoogleGenerativeAI  from langchain\_core.messages import HumanMessage  import gradio as gr  import getpass  import io  # Get Google API key securely  GOOGLE\_API\_KEY = getpass.getpass("Enter your Google API key: ")  # Initialize Gemini 2.0 LLM  llm = ChatGoogleGenerativeAI(  model="gemini-2.0-flash-exp",  temperature=0.8,  api\_key=GOOGLE\_API\_KEY,  max\_tokens=512,  timeout=30,  max\_retries=2,  )  print("Gemini LLM is ready.")  # Get similar words from Gemini  def get\_similar\_words(seed):  prompt =(  f"Give me 5 English words that are semantically or creatively similar to '{seed}'. "  f"Return the words as a comma separated list without numbers or explanations."  )  response = llm.invoke([HumanMessage(content=prompt)])  return [word.strip() for word in response.content.split(',') if word.strip()]  def create\_paragraph(seed, words):  word\_list = ', '.join(words)  prompt = (  f"Write a short, creative paragraph using the words '{seed}' and the following related words: "  f"{word\_list}. The paragraph should be imaginative and meaningful."  )  response = llm.invoke([HumanMessage(content=prompt)])  return response.content  def generate\_paragraphs(seed\_word):  try:  seed\_word = seed\_word.strip()  if not seed\_word:  return "Please enter a valid seed word."  similar\_words = get\_similar\_words(seed\_word)  if len(similar\_words) < 3:  return "Could not find similar words. Try a different seed word."  output\_text = f"Seed Word: {seed\_word}\nSimilar Words: {', '.join(similar\_words)}\n\n"  for i in range(1, 4):  paragraph=create\_paragraph(seed\_word, similar\_words) or f"(Variation{i})Could not generate paragraph."  output\_text += f"--- Variation {i} ---\n{paragraph.strip()}\n\n"  return output\_text  except Exception as e:  return f"Error: {str(e)}"  gr.Interface(  fn=generate\_paragraphs,  inputs=gr.Textbox(label="Enter a Seed Word"),  outputs=gr.Markdown(label="Generated Paragraphs"),  title="Creative Writer",  description="Enter a seed word. This app will find similar words using Gemini and generate 3 creative  paragraph variations.",  theme="default",  ).launch(debug=False) | 6) from google.colab import drive  drive.mount('/content/drive')  import os  cache\_dir = "/content/drive/MyDrive/transformers\_cache"  os.makedirs(cache\_dir, exist\_ok=True)  os.environ['TRANSFORMERS\_CACHE'] = cache\_dir  !pip install transformers pandas –quiet  import pandas as pd  from transformers import pipeline, AutoTokenizer, AutoModelForSequenceClassification  model\_name = "distilbert-base-uncased-finetuned-sst-2-english"  tokenizer = AutoTokenizer.from\_pretrained(model\_name, cache\_dir=cache\_dir)  model = AutoModelForSequenceClassification.from\_pretrained(model\_name, cache\_dir=cache\_dir)  # Create the sentiment analysis pipeline  sentiment\_pipeline = pipeline(  "sentiment-analysis",  model=model,  tokenizer=tokenizer  )  sample\_reviews = [  "I absolutely loved this product, it exceeded my expectations!",  "Great experience, the product quality and delivery were excellent.",  "Highly recommended! I’m very happy with the purchase.",  "The design is sleek and the features work perfectly.",  "Terrible experience. The product stopped working in two days.",  "Not worth the money — very disappointed with the quality."  ]  sentiment\_results = sentiment\_pipeline(sample\_reviews)  df\_results = pd.DataFrame({  "Review": sample\_reviews,  "Sentiment": [result["label"] for result in sentiment\_results],  "Confidence Score": [result["score"] for result in sentiment\_results]  })  df\_results  num\_positive = sum(1 for res in sentiment\_results if res["label"] == "POSITIVE")  num\_negative = sum(1 for res in sentiment\_results if res["label"] == "NEGATIVE")  total\_reviews = len(sentiment\_results)  positive\_percentage = (num\_positive / total\_reviews) \* 100  negative\_percentage = (num\_negative / total\_reviews) \* 100  # Determine overall sentiment  if num\_positive > num\_negative:  overall\_sentiment = "Positive"  recommendation = "We recommend this product based on the positive reviews."  elif num\_negative > num\_positive:  overall\_sentiment = "Negative"  recommendation = "We do not recommend this product based on the negative reviews."  else:  overall\_sentiment = "Mixed"  recommendation = "The reviews are mixed. Consider additional factors before deciding."  print("\n--- Overall Analysis ---")  print(f"Total Reviews Analyzed: {total\_reviews}")  print(f"Positive Reviews: {num\_positive} ({positive\_percentage:.1f}%)")  print(f"Negative Reviews: {num\_negative} ({negative\_percentage:.1f}%)")  print(f"Overall Sentiment: {overall\_sentiment}")  print(f"Recommendation: {recommendation}") |

|  |  |
| --- | --- |
| 7) !pip install transformers sentencepiece –quiet Step 2: Import libraries  from transformers import pipeline  from google.colab import files, drive  import os  Step 3: Mount Google Drive  drive.mount('/content/drive')  Step 4: Define model save path inside Google Drive  model\_dir = "/content/drive/MyDrive/bart\_summarizer"  Step 5: Load or download the model  if os.path.exists(model\_dir):  print("Loading model from Google Drive...")  summarizer = pipeline("summarization", model=model\_dir, tokenizer=model\_dir)  else:  print("Downloading model from Hugging Face for the first time...")  summarizer = pipeline("summarization", model="facebook/bart-large-cnn")  summarizer.model.save\_pretrained(model\_dir)  summarizer.tokenizer.save\_pretrained(model\_dir)  print("Model downloaded and saved to Google Drive")  Step 6: Upload a text file  print("\nPlease upload a text file with a long passage")  uploaded\_file = files.upload()  Step 7: Read the uploaded file  file\_name = list(uploaded\_file.keys())[0]  with open(file\_name, 'r') as file:  input\_text = file.read()  Step 8: Summarize the text file  print("\n Summarizing... please wait.")  summary = summarizer(input\_text, max\_length=150, min\_length=40, truncation=True)  summary\_text = summary[0]['summary\_text']  print("Summarization completed successfully")   Step 9: Display output  print("\n--- Original Text (First 500 characters) ---")  print(input\_text[:500] + "..." if len(input\_text) > 500 else input\_text)  print("\n--- Summarized Text ---")  print(summary\_text) | 8) Step 1: Install LangChain, Cohere, and LangChain-Cohere Plugin  !pip install langchain cohere langchain-community langchain-cohere –quiet  Step 2: Import required Python libraries  from langchain import PromptTemplate  from langchain\_community.llms import Cohere  from google.colab import drive  import os  from getpass import getpass  Step 3: Mount your Google Drive to access text files  drive.mount('/content/drive')  Step 4: Load and Read the text file content from Google Drive  file\_path = "/content/drive/MyDrive/input.txt"  with open(file\_path, 'r') as file:  document\_text = file.read()  print("Document loaded, length:", len(document\_text), "characters")  Step 5: Set your Cohere API key securely (input will be hidden)  os.environ["COHERE\_API\_KEY"] = getpass("Enter your Cohere API key: ")  print("Cohere API key configured.")  Step 6: Define the prompt format using LangChain's PromptTemplate  template = """  Summarize the following document in three bullet points highlighting the key ideas:  {content}  Bullet Point Summary:  """  prompt = PromptTemplate(input\_variables=["content"], template=template)  Step 7: Use the Cohere LLM to generate a response from the formatted prompt  llm = Cohere(max\_tokens=150, temperature=0) # temperature=0 for consistent output  formatted\_prompt = prompt.format(content=document\_text)  response = llm(formatted\_prompt)  # Displaying the formatted response  print("Formatted Output:\n", response) |

|  |  |
| --- | --- |
| 9) Step 1: Installing Required Libraries for LangChain and Pydantic Integration  !pip install -U langchain-cohere langchain pydantic –quiet  Step 2: Importing Core Components for Prompting and Data Modeling  from langchain.prompts import PromptTemplate  from langchain.chains import LLMChain  from pydantic import BaseModel  Step 3: Defining Output Structure with Pydantic and Creating a Prompt Template for the LLM  # Define Pydantic schema for the desired output  class InstitutionDetails (BaseModel):  founder: str  founded: str  branches: int  employees: int  summary: str  # Define the prompt template  prompt\_template = """  Given the name of an institution, extract the following details from Wikipedia:  1. Started By  2. Founded in the Year  3. Current Programs of the institution  4. How many employees work in it  5. A brief summary of the institution  Institution: {institution\_name}  """  Step 4: Setting the Cohere API Key Securely Using getpass and Environment Variables  import getpass  import os  os.environ["COHERE\_API\_KEY"] = getpass.getpass("Enter API key for Cohere: ")  Step 5: Creating the Language Model Chain with Cohere and Prompt Template  from langchain\_cohere import ChatCohere  model = ChatCohere(model="command-r7b-12-2024")  prompt = PromptTemplate(input\_variables=["institution\_name"],  template=prompt\_template)  chain = LLMChain(llm=model, prompt=prompt)  Step 6: Function to Interact with the LLM and Retrieve Institution Details  def fetch\_institution\_details(institution\_name: str):  result = chain.run(institution\_name=institution\_name)  return result  Step 7: Take institution name input from the user and Displaying Institution Information via LLM  institution\_name = input("Enter the institution name: ")  institution\_details = fetch\_institution\_details(institution\_name)  print(institution\_details) | 10) Step 1: Install Required Libraries.These help us read PDF, split it into chunks, embed it for search, and  interact with Gemini AI  !pip install -U langchain langchain-community faiss-cpu sentence-transformers PyPDF2 –quiet  !pip install -U google-generativeai==0.8.5 google-ai-generativelanguage==0.6.15 langchain-google-  genai -q  !pip install -q pypdf  Step 2: Import Libraries  import os  from getpass import getpass  from google.colab import files  from langchain\_community.document\_loaders import PyPDFLoader  from langchain.text\_splitter import CharacterTextSplitter  from langchain\_community.vectorstores import FAISS  from langchain\_community.embeddings import HuggingFaceEmbeddings  from langchain.chains import RetrievalQA  from langchain\_google\_genai import ChatGoogleGenerativeAI  Step 3: Read the Gemini API Key  print("Enter your Gemini API key.")  print("You can get API Key from https://aistudio.google.com/app/apikey")  api\_key = getpass("Enter your Gemini API Key: ")  os.environ["GOOGLE\_API\_KEY"] = api\_key  print("API key stored securely.")  Step 4: Upload Indian Penal Code PDF  print("\n Please upload the 'IndianPenalCode.pdf' file (or any IPC document).")  uploaded = files.upload()  pdf\_file = list(uploaded.keys())[0] # Get uploaded filename  print(f"PDF '{pdf\_file}' uploaded successfully.")  Step 5: Load and Split the PDF into Chunks  print("\n Reading and splitting the document...")  loader = PyPDFLoader(pdf\_file)  documents = loader.load()  text\_splitter = CharacterTextSplitter(chunk\_size=1000, chunk\_overlap=200)  docs = text\_splitter.split\_documents(documents)  print("Document split Completed.")  Step 6: Generate Embeddings and Store in FAISS Index  print("Creating embeddings for search...")  embedding\_model = HuggingFaceEmbeddings()  vectorstore = FAISS.from\_documents(docs, embedding\_model)  print("FAISS vector store created.")  Step 7: Load Gemini 2.0 Flash Model  llm = ChatGoogleGenerativeAI(model="gemini-2.0-flash-exp", temperature=0.3)  print("Gemini model is ready")  Step 8: Create RetrievalQA Chain  qa\_chain = RetrievalQA.from\_chain\_type(  llm=llm,  chain\_type="stuff",  retriever=vectorstore.as\_retriever()  )  print("Chatbot chain is configured.")  Step 9: Ask Questions to the Chatbot  print("\n Chatbot is ready You can now ask questions about the Indian Penal Code.")  print("Type exit to stop \n")  while True:  query = input("Ask a question: ")  if query.lower() in ['exit', 'quit']:  print("Thank you for using the IPC Chatbot")  break  response = qa\_chain.run(query)  print(f"Answer: {response}\n") |