**P-1**

**# Paths to the GloVe file and output Word2Vec file**

glove\_input\_file = "/content/glove.6B.100d.txt" # Path to GloVe file

word2vec\_output\_file = "/content/glove.6B.100d.word2vec.txt" # Output file in Word2Vec format

**# Convert GloVe format to Word2Vec format**

glove2word2vec(glove\_input\_file, word2vec\_output\_file)

# Load the converted Word2Vec model

model = KeyedVectors.load\_word2vec\_format(word2vec\_output\_file, binary=False)

**# Test the loaded model**

print(model.most\_similar("king"))

Example 1: Find Similar Words

similar\_to\_mysore = model.similar\_by\_vector(model['mysore'], topn=5)

print(f"Words similar to 'mysore': {similar\_to\_mysore}")

**Example 2: Gender Analogy (king - man + woman = queen)**

**# Perform vector arithmetic**

result\_vector\_1 = model['actor'] - model['man'] + model['woman']

result\_1 = model.similar\_by\_vector(result\_vector\_1, topn=1)

print(f"'actor - man + woman' = {result\_1}")

**Example 3: Country-City Relationship (India - Delhi + Bangalore)**

result\_vector\_2 = model['india'] - model['delhi'] + model['washington']

result\_2 = model.similar\_by\_vector(result\_vector\_2, topn=3)

**Perform Arithmetic Operations**

scaled\_vector = model['hotel'] \* 2 # Scales the 'king' vector by a factor of 2

result\_2 = model.similar\_by\_vector(scaled\_vector, topn=3)

result\_2

**Example 2: Normalizing Vectors**

import numpy as np

normalized\_vector = model['fish'] / np.linalg.norm(model['fish'])

result\_2 = model.similar\_by\_vector(normalized\_vector, topn=3)

result\_2

**Example 3: Averaging Vectors**

average\_vector = (model['king'] + model['woman'] + model['man']) / 3

result\_2 = model.similar\_by\_vector(average\_vector, topn=3)

result\_2

**Model Comparision**

glove\_input\_file = "/content/glove.6B.50d.txt" # Path to GloVe file

word2vec\_output\_file = "/content/glove.6B.50d.word2vec.txt" # Output file in Word2Vec format

glove2word2vec(glove\_input\_file, word2vec\_output\_file)

model\_50d = KeyedVectors.load\_word2vec\_format(word2vec\_output\_file, binary=False)

glove\_input\_file = "/content/glove.6B.100d.txt" # Path to GloVe file

word2vec\_output\_file = "/content/glove.6B.100d.word2vec.txt" # Output file in Word2Vec format

glove2word2vec(glove\_input\_file, word2vec\_output\_file)

model\_100d = KeyedVectors.load\_word2vec\_format(word2vec\_output\_file, binary=False)

**Calculate similarity between two words**

word1 = "hospital"

word2 = "doctor"

similarity\_50d = model\_50d.similarity(word1, word2)

similarity\_100d = model\_100d.similarity(word1, word2)

print(f"Similarity (50d) between '{word1}' and '{word2}': {similarity\_50d:.4f}")

print(f"Similarity (100d) between '{word1}' and '{word2}': {similarity\_100d:.4f}")

**Calculate distance between two words**

distance\_50d = model\_50d.distance(word1, word2)

distance\_100d = model\_100d.distance(word1, word2)

print(f"Distance (50d) between '{word1}' and '{word2}': {distance\_50d:.4f}")

print(f"Distance (100d) between '{word1}' and '{word2}': {distance\_100d:.4f}")

**P-2**

model\_100d = KeyedVectors.load\_word2vec\_format("/content/glove.6B.100d.word2vec.txt", binary=False,limit=500000)

words = ['football', 'soccer', 'basketball', 'tennis','engineer','information', 'baseball', 'coach', 'goal', 'player', 'referee', 'team']

word\_vectors = np.array([model\_100d[word] for word in words])

pca = PCA(n\_components=2)

pca\_result = pca.fit\_transform(word\_vectors)

**# Plotting the words in 2D space**

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

for i, word in enumerate(words):

plt.scatter(pca\_result[i, 0], pca\_result[i, 1])

plt.text(pca\_result[i, 0] + 0.02, pca\_result[i, 1], word, fontsize=12)

plt.title("PCA Visualization of Sports-related Word Embeddings (100d)")

plt.xlabel("PCA Dimension 1")

plt.ylabel("PCA Dimension 2")

plt.show()

def get\_similar\_words(word, model, topn=5):

similar\_words = model.similar\_by\_word(word, topn=topn)

return similar\_words

**Example: Get 5 words similar to "football"**

similar\_words\_football = get\_similar\_words('football', model\_100d, topn=5)

print(f"Words similar to 'football': {similar\_words\_football}")

Select the words you want to print embeddings for

words\_to\_print = ['football', 'soccer']

for word in words\_to\_print:

if word in model\_100d:

print(f"Vector embedding for '{word}':\n{model\_100d[word]}\n")

else:

print(f"Word '{word}' not found in the embeddings model.")

**P-3**

legal\_corpus = [ **Write any 6 sentence** ]

legal\_corpus = [ Same 6 sentence ]

**# Preprocess the corpus**

tokenized\_corpus = [simple\_preprocess(sentence) for sentence in legal\_corpus]

legal\_word2vec = Word2Vec(

sentences=tokenized\_corpus,

vector\_size=50,

window=3,

min\_count=1,

sg=1,

epochs=100

)

legal\_word2vec.save("legal\_word2vec.model")

word = "lawyer"

if word in legal\_word2vec.wv:

print(f"Vector embedding for '{word}':\n{legal\_word2vec.wv[word]}\n")

else:

print(f"Word '{word}' not found in the Word2Vec model.")

words\_to\_visualize = ["court", "plaintiff", "defendant", "agreement", "lawyer", "evidence", "contract", "settlement", "jury", "damages"]

word\_vectors = [legal\_word2vec.wv[word] for word in words\_to\_visualize]

**# Dimensionality reduction**

pca = PCA(n\_components=2)

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

reduced\_vectors

**# Dimensionality reduction**

pca = PCA(n\_components=2)

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

reduced\_vectors

**# Find similar words**

similar\_words = legal\_word2vec.wv.most\_similar("lawyer", topn=5)

print(f"Words similar to 'lawyer': {similar\_words}")

enhanced\_corpus = [ Repeat those 6 sentence ]

**# Preprocess the corpus**

tokenized\_corpus = [simple\_preprocess(sentence) for sentence in enhanced\_corpus]

tokenized\_corpus

**P-4**

**# Paths to the GloVe file and output Word2Vec file**

glove\_input\_file = "/content/glove.6B.100d.txt" # Path to GloVe file

word2vec\_output\_file = "/content/glove.6B.100d.word2vec.txt" # Output file in Word2Vec format

**# Convert GloVe format to Word2Vec format**

glove2word2vec(glove\_input\_file, word2vec\_output\_file)

**# Load the converted Word2Vec model**

model = KeyedVectors.load\_word2vec\_format(word2vec\_output\_file, binary=False)

**# Test the loaded model**

print(model.most\_similar("king"))

# Define the original medical prompt

original\_prompt = "Explain the importance of vaccinations in healthcare."

**# Define key terms extracted from the original prompt**

key\_terms = ["vaccinations", "healthcare"]

**# Initialize an empty list to store similar terms**

similar\_terms = []

**# Loop through each key term to find similar words**

**for term in key\_terms:**

if term in model.key\_to\_index:

similar\_terms.extend({word for word, \_ in model.most\_similar(term, topn=3)})

**# Enrich the original prompt with the retrieved similar terms**

if similar\_terms:

**# If similar terms were found, create an enriched prompt by appending**

**# "Consider aspects like: " followed by a comma-separated string of similar terms.**

enriched\_prompt = f"{original\_prompt} Consider aspects like: {', '.join(similar\_terms)}."

else:

**# If no similar terms were found, the enriched prompt is the same as the original prompt.**

enriched\_prompt = original\_prompt

**# Output the original and enriched prompts**

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: ")

llm = ChatGoogleGenerativeAI(

model="gemini-2.0-flash-exp",

temperature=0,

api\_key=GOOGLE\_API\_KEY,

max\_tokens=256,

timeout=None,

max\_retries=2,

# other params...

)

llm.invoke("Hi")

print(llm.invoke(enriched\_prompt).content)

**P-5**

**# Load a pretrained SentenceTransformer model**

model = SentenceTransformer('all-MiniLM-L6-v2')

**# Define an expanded finance-related corpus**

corpus = [ Any 7 sentence ]

**# Encode the corpus into embeddings**

corpus\_embeddings = model.encode(corpus, convert\_to\_tensor=True)

corpus\_embeddings

**# Function to generate a story using contextual embeddings**

def generate\_paragraph(seed\_word, corpus, corpus\_embeddings, model, top\_n=5):

**# Encode the seed word as a sentence**

seed\_sentence = f"Tell me more about {seed\_word} in finance."

seed\_embedding = model.encode(seed\_sentence, convert\_to\_tensor=True)

**# Find the most similar sentences in the corpus to the seed sentence**

similarities = util.pytorch\_cos\_sim(seed\_embedding, corpus\_embeddings)[0]

top\_results = similarities.topk(top\_n)

print('top\_results:',top\_results)

**# Construct a more coherent story using the most similar sentences**

story = f"The topic of '{seed\_word}' is crucial in the finance industry. "

for idx in top\_results.indices:

similar\_sentence = corpus[idx]

story += f"{similar\_sentence} "

story += f"These concepts highlight the importance of {seed\_word} in understanding financial markets and investment strategies."

**# Example usage**

seed\_word = "bonds"

story = generate\_paragraph(seed\_word, corpus, corpus\_embeddings, model, top\_n=5)

print(story)