!pip install gensim

#Gensim: A Python library for NLP and word embeddings.

from gensim.scripts.glove2word2vec import glove2word2vec

from gensim.models import KeyedVectors

# 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"))

#GloVe embeddings are converted to Word2Vec format for compatibility with libraries like

Gensim, which require the Word2Vec format for efficient vector operations and model

functionality.

Output: [('prince', 0.7682328820228577), ('queen', 0.7507690787315369),

('son', 0.7020888328552246), ('brother', 0.6985775232315063),

('monarch', 0.6977890729904175), ('throne', 0.6919989585876465),

('kingdom', 0.6811409592628479), ('father', 0.6802029013633728),

('emperor', 0.6712858080863953), ('ii', 0.6676074266433716)]

Explore Word Relationships

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}")

Output: Words similar to 'mysore': [('mysore', 1.0), ('cochin',

0.6752076148986816), ('hyderabad', 0.6592637896537781), ('jaipur',

0.6591896414756775), ('perak', 0.6516631245613098)]

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

# Perform vector arithmetic

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

# Find the most similar word

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

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

Output: 'actor - man + woman' = [('actress', 0.9160683155059814)]

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

# Perform vector arithmetic

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

# Find the most similar word

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

print(f"'India - Delhi + Washington' = {result\_2}")

Output: 'India - Delhi + Washington' = [('states', 0.8375228643417358),

('united', 0.8281229734420776), ('washington', 0.8155243396759033)]

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

[('hotel', 1.0),

('hotels', 0.7933705449104309),

('restaurant', 0.7762866020202637)]

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

[('fish', 1.0), ('shrimp', 0.7793381810188293), ('salmon', 0.760814368724823)]

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

[('man', 0.9197071194648743),

('woman', 0.8637868165969849),

('father', 0.8270207047462463)]

Model Comparision

# Paths to the GloVe file and output Word2Vec file

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

# Convert GloVe format to Word2Vec format

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

# Load the converted Word2Vec model

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

binary=False)

# 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\_100d = KeyedVectors.load\_word2vec\_format(word2vec\_output\_file,

binary=False)

Calculate similarity between two words

word1 = "hospital"

word2 = "doctor"

# Similarity in 50d

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

# Similarity in 100d

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

# Results

print(f"Similarity (50d) between '{word1}' and '{word2}':

{similarity\_50d:.4f}")

print(f"Similarity (100d) between '{word1}' and '{word2}':

{similarity\_100d:.4f}")

Output : Similarity (50d) between 'hospital' and 'doctor': 0.6724

Similarity (100d) between 'hospital' and 'doctor': 0.6901

Calculate distance between two words

# Calculate distance between two words

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

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

# Results

print(f"Distance (50d) between '{word1}' and '{word2}':

{distance\_50d:.4f}")

print(f"Distance (100d) between '{word1}' and '{word2}':

{distance\_100d:.4f}")

Distance (50d) between 'hospital' and 'doctor': 0.3276

Distance (100d) between 'hospital' and 'doctor': 0.3099

Pro2:

import numpy as np

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.manifold import TSNE

from gensim.models import KeyedVectors

# Load pre-trained GloVe embeddings (100d model)

model\_100d =

KeyedVectors.load\_word2vec\_format("/content/glove.6B.100d.word2vec.txt"

, binary=False,limit=500000)

# Select 10 words from a specific domain (sports) # Included other

words to show how embeddings are different

words = ['football', 'soccer', 'basketball',

'tennis','engineer','information', 'baseball', 'coach', 'goal',

'player', 'referee', 'team']

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

# Dimensionality reduction using PCA

# Using PCA to reduce to 2D for visualization

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()

# 5 Semantically Similar Words Generator Function

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']

# Print their embeddings

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.")

pro3

from gensim.models import Word2Vec

from gensim.utils import simple\_preprocess

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

legal\_corpus = [

"The court ruled in favor of the plaintiff.",

"The defendant was found guilty of negligence.",

"A breach of contract case was filed.",

"The agreement between parties must be honored.",

"The lawyer presented compelling evidence.",

"Legal documents must be drafted carefully.",

"The jury deliberated for several hours.",

"A settlement was reached between the parties.",

"The plaintiff claimed damages for losses incurred.",

"The contract outlined the obligations of both parties."

]

# Example legal corpus

legal\_corpus = [

"The court ruled in favor of the plaintiff.",

"The defendant was found guilty of negligence.",

"A breach of contract case was filed.",

"The agreement between parties must be honored.",

"The lawyer presented compelling evidence.",

"Legal documents must be drafted carefully.",

"The jury deliberated for several hours.",

"A settlement was reached between the parties.",

"The plaintiff claimed damages for losses incurred.",

"The contract outlined the obligations of both parties."

]

# Preprocess the corpus

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

legal\_corpus]

# Train the Word2Vec model

legal\_word2vec = Word2Vec(

sentences=tokenized\_corpus,

vector\_size=50, # Embedding dimension

window=3, # Context window size

min\_count=1, # Minimum word frequency

sg=1, # Skip-gram model

epochs=100 # Training epochs

)

# Save the model for later use

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

# Analyze embeddings: Display vector for a specific word

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]

word\_vectors

pca = PCA(n\_components=2)

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

reduced\_vectors

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

for i, word in enumerate(words\_to\_visualize):

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

plt.text(reduced\_vectors[i, 0] + 0.002, reduced\_vectors[i, 1],

word, fontsize=12)

plt.title("PCA Visualization of Legal Word Embeddings")

plt.xlabel("PCA Dimension 1")

plt.ylabel("PCA Dimension 2")

plt.show()

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

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

Example: Legal and Medical / Healthcare Corpus

from gensim.models import Word2Vec

from gensim.utils import simple\_preprocess

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

# Enhanced legal and medical corpus

enhanced\_corpus = [

# Legal domain

"The court ordered the immediate release of the detained individual

due to lack of evidence.",

"A new amendment was introduced to ensure the protection of

intellectual property rights.",

"The defendant pleaded not guilty, citing an alibi supported by

credible witnesses.",

"The plaintiff accused the company of violating environmental

regulations.",

"A settlement agreement was reached through arbitration, avoiding a

lengthy trial.",

"The legal team presented a compelling argument to overturn the

previous judgment.",

"Contractual obligations must be fulfilled unless waived by mutual

consent.",

"The jury found the accused guilty of fraud and embezzlement.",

"The appeal was dismissed as the evidence presented was deemed

inadmissible.",

"The attorney emphasized the importance of adhering to

constitutional rights.",

# Medical domain

"The patient was admitted to the emergency department with severe

chest pain.",

"The surgeon successfully performed a minimally invasive procedure

to remove the tumor.",

"Clinical trials showed significant improvement in patients treated

with the experimental drug.",

"Regular screening is essential for early detection of chronic

illnesses such as diabetes.",

"The doctor recommended physical therapy to improve mobility after

surgery.",

"The hospital implemented stringent protocols to prevent the spread

of infectious diseases.",

"The nurse monitored the patient's vital signs hourly to ensure

stability.",

"Vaccination campaigns have drastically reduced the prevalence of

polio worldwide.",

"The radiologist identified a small abnormality in the CT scan

requiring further investigation.",

"Proper nutrition and exercise are vital components of a healthy

lifestyle."

]

# Preprocess the corpus

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

enhanced\_corpus]

tokenized\_corpus

# Train Word2Vec

domain\_word2vec = Word2Vec(

sentences=tokenized\_corpus,

vector\_size=100, # Higher embedding dimension for better

representation

window=5,

# Wider context window

min\_count=1,

# Include all words

sg=1,

# Skip-gram model

epochs=150

# More training iterations

)

# Save the model

domain\_word2vec.save("enhanced\_domain\_word2vec.model")

# Analyze embeddings: Get vectors for specific words

words\_to\_analyze = ["court", "plaintiff", "doctor", "patient",

"guilty", "surgery"]

for word in words\_to\_analyze:

if word in domain\_word2vec.wv:

print(f"Vector embedding for

'{word}':\n{domain\_word2vec.wv[word]}\n")

else:

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

pca = PCA(n\_components=2)

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

reduced\_vectors

plt.figure(figsize=(12, 8))

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

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

plt.text(reduced\_vectors[i, 0] + 0.002, reduced\_vectors[i, 1],

word, fontsize=12)

plt.title("PCA Visualization of Legal and Medical Word Embeddings")

plt.xlabel("PCA Dimension 1")

plt.ylabel("PCA Dimension 2")

pro4

pip install transformers –U

from gensim.scripts.glove2word2vec import glove2word2vec

from gensim.models import KeyedVectors

# 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"))

Output:

[('prince', 0.7682328820228577), ('queen', 0.7507690787315369), ('son',

0.7020888328552246), ('brother', 0.6985775232315063), ('monarch',

0.6977890729904175), ('throne', 0.6919989585876465), ('kingdom',

0.6811409592628479), ('father', 0.6802029013633728), ('emperor',

0.6712858080863953), ('ii', 0.6676074266433716)]

# 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:

# Check if the key term exists in the vocabulary of the 'model'

(word embedding model)

# Assuming 'model.key\_to\_index' is a way to check for term

existence in the model's vocabulary

if term in model.key\_to\_index:

# If the term exists, find the top 3 most similar words using

'model.most\_similar(term, topn=3)'

# and extend the 'similar\_terms' list with these words.

# Assuming 'model.most\_similar' returns a list of tuples, where

each tuple is (word, similarity\_score)

# We are extracting only the 'word' part using a set

comprehension for potential deduplication.

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)

pro5

!pip install sentence\_transformers

!pip install langchain-huggingface

!pip install tf-keras --user

!pip install numpy==1.24.4 --user

from sentence\_transformers import SentenceTransformer, util

# Load a pretrained SentenceTransformer model

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

# Define an expanded finance-related corpus

corpus = [

"The stock market saw significant gains today, driven by strong

earnings reports.",

"Investing in diversified portfolios helps mitigate risk and

maximize returns.",

"The Federal Reserve's decision to raise interest rates could

impact market liquidity.",

"Cryptocurrency has become an increasingly popular asset class

among investors.",

"Financial analysts predict that the global economy will face

challenges in the coming years.",

"Bonds are considered a safer investment option compared to

stocks.",

"Banks are adopting blockchain technology to improve the efficiency

of financial transactions.",

"The economic impact of the pandemic has been severe, but recovery

is underway.",

"Inflation rates have been rising steadily, leading to higher costs

for consumers.",

"Corporate governance and transparency are crucial for investor

confidence.",

"The real estate market is experiencing a boom as demand outstrips

supply in many areas.",

"Investors should be aware of market volatility and adjust their

strategies accordingly.",

"Diversification is a key principle in reducing risk in investment

portfolios.",

"Hedge funds use complex strategies to generate high returns, often

with higher risks.",

"Stock buybacks are often seen as a sign of confidence by corporate

executives."

]

# Encode the corpus into embeddings

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

corpus\_embeddings

Output:

tensor([[ 0.0129, 0.0182, -0.0129, ..., -0.0351, -0.0190, 0.0443], [

0.0329, 0.0204, -0.0503, ..., -0.0383, -0.0037, 0.0154], [-0.0168,

0.0174, -0.0506, ..., -0.0439, 0.0390, -0.0251], ..., [ 0.0668, 0.0304, -0.0115, ..., -0.0700, -0.0742, -0.0177], [-0.0069, -0.0231, -0.0392,

..., -0.0815, 0.0679, 0.0207], [-0.0347, -0.0332, 0.0320, ..., -0.0874, -0.0046, 0.0356]])

# 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."

return story

# Example usage

seed\_word = "bonds"

story = generate\_paragraph(seed\_word, corpus, corpus\_embeddings, model,

top\_n=5)

print(story)