RAG_Rothman: Chapter 1: Overview

Saturday 28th December, 2024 at 11:43

#Introducing Naive, Advanced, and Modular RAG

This notebook introduces Naïve, Advanced, and Modular RAG through basic educational examples.

The Naïve, Advanced and modular RAG techniques offer flexibility in selecting retrieval strategies, allowing adaptation to various tasks and data characteristics.

Summary

Part 1: Foundations and Basic Implementation

1.Environment setup for OpenAI API integration
2.Generator function using GPT models
3.Dataetup with a list of documents (db_records)
4.Query(user request)

Part 2: Advanced Techniques and Evaluation

1.Retrieval metrics
2.Naive RAG
3.Advanced RAG
4.Modular RAG Retriever

1 Part 1: Foundations and Basic Implementation

2 1.The Environment

[1]:
```
#!pip install openai==1.40.3
```

[2]:
```
#API Key
#Store you key in a file and read it(you can type it directly in the notebook␣
↪but it will be visible for somebody next to you)
#from google.colab import drive
#drive.mount('/content/drive')
```

[7]:
```
#f = open("drive/MyDrive/files/api_key.txt", "r")
#API_KEY=f.readline().strip()
#f.close()
```

```
#The OpenAI Key
import os
from dotenv import load_dotenv
import openai

# Load API Key
dotenv_path = 'D:/AdvancedR/knowbankedu/openai/.env'
load_dotenv(dotenv_path)
# OpenAI API Key
openai.api_key = os.getenv("OPENAI_API_KEY")
```

3 2.The Generator

[8]:
```
import openai
from openai import OpenAI

client = OpenAI()
gptmodel="gpt-4o"

def call_llm_with_full_text(itext):
    # Join all lines to form a single string
    text_input = '\n'.join(itext)
    prompt = f"Please elaborate on the following content:\n{text_input}"

    try:
      response = client.chat.completions.create(
         model=gptmodel,
         messages=[
            {"role": "system", "content": "You are an expert Natural Language␣
 ↪Processing exercise expert."},
            {"role": "assistant", "content": "1.You can explain read the input␣
 ↪and answer in detail"},
            {"role": "user", "content": prompt}
         ],
         temperature=0.1  # Add the temperature parameter here and other␣
 ↪parameters you need
       )
      return response.choices[0].message.content.strip()
   except Exception as e:
       return str(e)
```

3.1 Formatted response

[9]:
```
import textwrap

def print_formatted_response(response):
    # Define the width for wrapping the text
    wrapper = textwrap.TextWrapper(width=80)  # Set to 80 columns wide, but␣
↪adjust as needed
    wrapped_text = wrapper.fill(text=response)

    # Print the formatted response with a header and footer
    print("Response:")
    print("---------------")
    print(wrapped_text)
    print("---------------\n")
```

3.The Data

[10]:
```
db_records = [
    "Retrieval Augmented Generation (RAG) represents a sophisticated hybrid␣
↪approach in the field of artificial intelligence, particularly within the␣
↪realm of natural language processing (NLP).",
    "It innovatively combines the capabilities of neural network-based language␣
↪models with retrieval systems to enhance the generation of text, making it␣
↪more accurate, informative, and contextually relevant.",
    "This methodology leverages the strengths of both generative and retrieval␣
↪architectures to tackle complex tasks that require not only linguistic fluency␣
↪but also factual correctness and depth of knowledge.",
    "At the core of Retrieval Augmented Generation (RAG) is a generative model,␣
↪typically a transformer-based neural network, similar to those used in models␣
↪like GPT (Generative Pre-trained Transformer) or BERT (Bidirectional Encoder␣
↪Representations from Transformers).",
    "This component is responsible for producing coherent and contextually␣
↪appropriate language outputs based on a mixture of input prompts and␣
↪additional information fetched by the retrieval component.",
    "Complementing the language model is the retrieval system, which is usually␣
↪built on a database of documents or a corpus of texts.",
    "This system uses techniques from information retrieval to find and fetch␣
↪documents that are relevant to the input query or prompt.",
    "The mechanism of relevance determination can range from simple keyword␣
↪matching to more complex semantic search algorithms which interpret the␣
↪meaning behind the query to find the best matches.",
    "This component merges the outputs from the language model and the retrieval␣
↪system.",
    "It effectively synthesizes the raw data fetched by the retrieval system␣
↪into the generative process of the language model.",
```

```
   "The integrator ensures that the information from the retrieval system is␣
↪seamlessly incorporated into the final text output, enhancing the model's␣
↪ability to generate responses that are not only fluent and grammatically␣
↪correct but also rich in factual details and context-specific nuances.",
   "When a query or prompt is received, the system first processes it to␣
↪understand the requirement or the context.",
   "Based on the processed query, the retrieval system searches through its␣
↪database to find relevant documents or information snippets.",
   "This retrieval is guided by the similarity of content in the documents to␣
↪the query, which can be determined through various techniques like vector␣
↪embeddings or semantic similarity measures.",
   "The retrieved documents are then fed into the language model.",
   "In some implementations, this integration happens at the token level, where␣
↪the model can access and incorporate specific pieces of information from the␣
↪retrieved texts dynamically as it generates each part of the response.",
   "The language model, now augmented with direct access to retrieved␣
↪information, generates a response.",
   "This response is not only influenced by the training of the model but also␣
↪by the specific facts and details contained in the retrieved documents, making␣
↪it more tailored and accurate.",
   "By directly incorporating information from external sources, Retrieval␣
↪Augmented Generation (RAG) models can produce responses that are more factual␣
↪and relevant to the given query.",
   "This is particularly useful in domains like medical advice, technical␣
↪support, and other areas where precision and up-to-date knowledge are crucial.
↪",
   "Retrieval Augmented Generation (RAG) systems can dynamically adapt to new␣
↪information since they retrieve data in real-time from their databases.",
   "This allows them to remain current with the latest knowledge and trends␣
↪without needing frequent retraining.",
   "With access to a wide range of documents, Retrieval Augmented Generation␣
↪(RAG) systems can provide detailed and nuanced answers that a standalone␣
↪language model might not be capable of generating based solely on its␣
↪pre-trained knowledge.",
   "While Retrieval Augmented Generation (RAG) offers substantial benefits, it␣
↪also comes with its challenges.",
   "These include the complexity of integrating retrieval and generation␣
↪systems, the computational overhead associated with real-time data retrieval,␣
↪and the need for maintaining a large, up-to-date, and high-quality database of␣
↪retrievable texts.",
   "Furthermore, ensuring the relevance and accuracy of the retrieved␣
↪information remains a significant challenge, as does managing the potential␣
↪for introducing biases or errors from the external sources.",
```

```
    "In summary, Retrieval Augmented Generation represents a significant␣
 ↪advancement in the field of artificial intelligence, merging the best of␣
 ↪retrieval-based and generative technologies to create systems that not only␣
 ↪understand and generate natural language but also deeply comprehend and␣
 ↪utilize the vast amounts of information available in textual form.",
    "A RAG vector store is a database or dataset that contains vectorized data␣
 ↪points."
]
```

[11]:
```
import textwrap
paragraph = ' '.join(db_records)
wrapped_text = textwrap.fill(paragraph, width=80)
print(wrapped_text)
```

```
Retrieval Augmented Generation (RAG) represents a sophisticated hybrid approach
in the field of artificial intelligence, particularly within the realm of
natural language processing (NLP). It innovatively combines the capabilities of
neural network-based language models with retrieval systems to enhance the
generation of text, making it more accurate, informative, and contextually
relevant. This methodology leverages the strengths of both generative and
retrieval architectures to tackle complex tasks that require not only linguistic
fluency but also factual correctness and depth of knowledge. At the core of
Retrieval Augmented Generation (RAG) is a generative model, typically a
transformer-based neural network, similar to those used in models like GPT
(Generative Pre-trained Transformer) or BERT (Bidirectional Encoder
Representations from Transformers). This component is responsible for producing
coherent and contextually appropriate language outputs based on a mixture of
input prompts and additional information fetched by the retrieval component.
Complementing the language model is the retrieval system, which is usually built
on a database of documents or a corpus of texts. This system uses techniques
from information retrieval to find and fetch documents that are relevant to the
input query or prompt. The mechanism of relevance determination can range from
simple keyword matching to more complex semantic search algorithms which
interpret the meaning behind the query to find the best matches. This component
merges the outputs from the language model and the retrieval system. It
effectively synthesizes the raw data fetched by the retrieval system into the
generative process of the language model. The integrator ensures that the
information from the retrieval system is seamlessly incorporated into the final
text output, enhancing the model's ability to generate responses that are not
only fluent and grammatically correct but also rich in factual details and
context-specific nuances. When a query or prompt is received, the system first
processes it to understand the requirement or the context. Based on the
processed query, the retrieval system searches through its database to find
relevant documents or information snippets. This retrieval is guided by the
similarity of content in the documents to the query, which can be determined
through various techniques like vector embeddings or semantic similarity
measures. The retrieved documents are then fed into the language model. In some
implementations, this integration happens at the token level, where the model
```

can access and incorporate specific pieces of information from the retrieved texts dynamically as it generates each part of the response. The language model, now augmented with direct access to retrieved information, generates a response. This response is not only influenced by the training of the model but also by the specific facts and details contained in the retrieved documents, making it more tailored and accurate. By directly incorporating information from external sources, Retrieval Augmented Generation (RAG) models can produce responses that are more factual and relevant to the given query. This is particularly useful in domains like medical advice, technical support, and other areas where precision and up-to-date knowledge are crucial. Retrieval Augmented Generation (RAG) systems can dynamically adapt to new information since they retrieve data in real-time from their databases. This allows them to remain current with the latest knowledge and trends without needing frequent retraining. With access to a wide range of documents, Retrieval Augmented Generation (RAG) systems can provide detailed and nuanced answers that a standalone language model might not be capable of generating based solely on its pre-trained knowledge. While Retrieval Augmented Generation (RAG) offers substantial benefits, it also comes with its challenges. These include the complexity of integrating retrieval and generation systems, the computational overhead associated with real-time data retrieval, and the need for maintaining a large, up-to-date, and high-quality database of retrievable texts. Furthermore, ensuring the relevance and accuracy of the retrieved information remains a significant challenge, as does managing the potential for introducing biases or errors from the external sources. In summary, Retrieval Augmented Generation represents a significant advancement in the field of artificial intelligence, merging the best of retrieval-based and generative technologies to create systems that not only understand and generate natural language but also deeply comprehend and utilize the vast amounts of information available in textual form. A RAG vector store is a database or dataset that contains vectorized data points.

4 4.The Query

[12]:
```
query = "define a rag store"
```

Generation without augmentation

[13]:
```
# Call the function and print the result
llm_response = call_llm_with_full_text(query)
print_formatted_response(llm_response)
```

```
Response:
---------------
Certainly! The content you've provided seems to be a vertically arranged
sequence of letters that, when read horizontally, spells out "define a rag
store." Let's break down what this phrase could mean:  1. **Define**: This is a
verb that means to explain the meaning of a word or concept. In this context, it
suggests that we are being asked to provide a clear explanation or description
of what a "rag store" is.  2. **A Rag Store**: This term likely refers to a type
```

```
of retail establishment. Let's explore what a "rag store" might be:     -
**Rag**: Traditionally, a "rag" is a piece of old, often torn or worn-out cloth.
Rags are commonly used for cleaning or as material for crafting. In some
contexts, "rags" can also refer to old or second-hand clothing.     - **Store**:
This is a place where goods are sold to the public. It can range from a small
shop to a large retail outlet.  3. **Rag Store**: Combining these two words, a
"rag store" could be interpreted in a couple of ways:     - **Second-Hand
Clothing Store**: A store that sells used clothing, often at a lower price than
new items. These stores are popular for those looking for vintage or affordable
clothing options.     - **Fabric or Textile Store**: A store that specializes in
selling fabric remnants, scraps, or materials that can be used for sewing,
quilting, or crafting. These stores might cater to hobbyists or professionals
looking for unique or discounted fabric pieces.  In summary, a "rag store" is
likely a retail establishment that deals with either second-hand clothing or
fabric remnants. The exact nature of the store would depend on the specific
focus of its inventory and the needs of its customers.
---------------
```

5 Part 2: Advanced Techniques and Evaluation

6 1.Retrieval Metrics

6.1 Cosine Similarity

[14]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def calculate_cosine_similarity(text1, text2):
    vectorizer = TfidfVectorizer(
        stop_words='english',
        use_idf=True,
        norm='l2',
        ngram_range=(1, 2),  # Use unigrams and bigrams
        sublinear_tf=True,   # Apply sublinear TF scaling
        analyzer='word'      # You could also experiment with 'char' or␣
 ↪'char_wb' for character-level features
    )
    tfidf = vectorizer.fit_transform([text1, text2])
    similarity = cosine_similarity(tfidf[0:1], tfidf[1:2])
    return similarity[0][0]
```

6.2 Enhanced Similarity

[19]:
```
import spacy
import nltk
nltk.download('wordnet')
from nltk.corpus import wordnet
from collections import Counter
import numpy as np

# Load spaCy model
nlp = spacy.load("en_core_web_sm")

def get_synonyms(word):
    synonyms = set()
    for syn in wordnet.synsets(word):
        for lemma in syn.lemmas():
            synonyms.add(lemma.name())
    return synonyms

def preprocess_text(text):
    doc = nlp(text.lower())
    lemmatized_words = []
    for token in doc:
        if token.is_stop or token.is_punct:
            continue
        lemmatized_words.append(token.lemma_)
    return lemmatized_words

def expand_with_synonyms(words):
    expanded_words = words.copy()
    for word in words:
        expanded_words.extend(get_synonyms(word))
    return expanded_words

def calculate_enhanced_similarity(text1, text2):
    # Preprocess and tokenize texts
    words1 = preprocess_text(text1)
    words2 = preprocess_text(text2)

    # Expand with synonyms
    words1_expanded = expand_with_synonyms(words1)
    words2_expanded = expand_with_synonyms(words2)

    # Count word frequencies
    freq1 = Counter(words1_expanded)
    freq2 = Counter(words2_expanded)
```

```
    # Create a set of all unique words
    unique_words = set(freq1.keys()).union(set(freq2.keys()))

    # Create frequency vectors
    vector1 = [freq1[word] for word in unique_words]
    vector2 = [freq2[word] for word in unique_words]

    # Convert lists to numpy arrays
    vector1 = np.array(vector1)
    vector2 = np.array(vector2)

    # Calculate cosine similarity
    cosine_similarity = np.dot(vector1, vector2) / (np.linalg.norm(vector1) * np.
↪linalg.norm(vector2))

    return cosine_similarity
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\wb164718\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

7 2.Naive RAG

7.1 Keyword search and matching

[20]:
```
def find_best_match_keyword_search(query, db_records):
    best_score = 0
    best_record = None

    # Split the query into individual keywords
    query_keywords = set(query.lower().split())

    # Iterate through each record in db_records
    for record in db_records:
        # Split the record into keywords
        record_keywords = set(record.lower().split())

        # Calculate the number of common keywords
        common_keywords = query_keywords.intersection(record_keywords)
        current_score = len(common_keywords)

        # Update the best score and record if the current score is higher
        if current_score > best_score:
            best_score = current_score
            best_record = record

    return best_score, best_record
```

```
# Assuming 'query' and 'db_records' are defined in previous cells in your Colab notebook
best_keyword_score, best_matching_record = find_best_match_keyword_search(query, db_records)

print(f"Best Keyword Score: {best_keyword_score}")
print_formatted_response(best_matching_record)
```

```
Best Keyword Score: 3
Response:
---------------
A RAG vector store is a database or dataset that contains vectorized data
points.
---------------
```

7.2 Metrics

[21]:
```
# Cosine Similarity
score = calculate_cosine_similarity(query, best_matching_record)
print(f"Best Cosine Similarity Score: {score:.3f}")
```

```
Best Cosine Similarity Score: 0.126
```

[22]:
```
# Enhanced Similarity
response = best_matching_record
print(query,": ", response)
similarity_score = calculate_enhanced_similarity(query, response)
print(f"Enhanced Similarity:, {similarity_score:.3f}")
```

```
define a rag store :  A RAG vector store is a database or dataset that contains
vectorized data points.
Enhanced Similarity:, 0.642
```

7.3 Augmented input

[23]:
```
augmented_input=query+ ": "+ best_matching_record
```

[16]:
```
print_formatted_response(augmented_input)
```

```
Response:
---------------
define a rag store: A RAG vector store is a database or dataset that contains
vectorized data points.
---------------
```

7.4 Generation

[24]:
```
# Call the function and print the result
llm_response = call_llm_with_full_text(augmented_input)
print_formatted_response(llm_response)
```

```
Response:
---------------
An ARAG vector store, or more generally a vector store, is a specialized type of
database or dataset designed to store and manage vectorized data points. Here's
a detailed explanation of the concept:  1. **Vectorized Data Points**: In the
context of machine learning and data science, data is often represented in the
form of vectors. A vector is essentially an array of numbers that represents
data in a numerical format. This transformation from raw data to numerical
vectors is known as vectorization. For example, text data can be vectorized
using techniques like word embeddings (e.g., Word2Vec, GloVe) or sentence
embeddings (e.g., BERT).  2. **Purpose of a Vector Store**: The primary purpose
of a vector store is to efficiently store, retrieve, and manage these vectorized
data points. This is particularly useful in applications that require similarity
search, such as recommendation systems, information retrieval, and clustering.
3. **Similarity Search**: One of the key functionalities of a vector store is to
perform similarity searches. This involves finding vectors that are similar to a
given query vector. Similarity is often measured using distance metrics like
Euclidean distance or cosine similarity. This capability is crucial for tasks
like finding similar documents, images, or users based on their vector
representations.  4. **Scalability and Performance**: Vector stores are
optimized for handling large volumes of high-dimensional data. They are designed
to be scalable and performant, allowing for fast retrieval of similar vectors
even in datasets with millions of entries.  5. **Applications**: Vector stores
are widely used in various applications, including:    - **Recommendation
Systems**: To suggest products or content based on user preferences.    -
**Natural Language Processing (NLP)**: For tasks like semantic search and
document clustering.    - **Computer Vision**: To find similar images or
objects.    - **Anomaly Detection**: To identify outliers in data by comparing
vector representations.  6. **Examples of Vector Stores**: There are several
tools and platforms that provide vector storage capabilities, such as FAISS
(Facebook AI Similarity Search), Annoy (Approximate Nearest Neighbors Oh Yeah),
and Elasticsearch with its vector search capabilities.  In summary, an ARAG
vector store is a crucial component in modern data-driven applications, enabling
efficient handling and retrieval of vectorized data for various analytical and
predictive tasks.
---------------
```

8 3.Advanced RAG

8.1 3.1.Vector search

8.1.1 Search function

[28]:
```
def find_best_match(text_input, records):
    best_score = 0
    best_record = None
    for record in records:
        current_score = calculate_cosine_similarity(text_input, record)
        if current_score > best_score:
            best_score = current_score
            best_record = record
    return best_score, best_record
```

[29]:
```
best_similarity_score, best_matching_record = find_best_match(query, db_records)
```

[30]:
```
print_formatted_response(best_matching_record)
```

```
Response:
---------------
A RAG vector store is a database or dataset that contains vectorized data
points.
---------------
```

8.1.2 Metrics

[31]:
```
print(f"Best Cosine Similarity Score: {best_similarity_score:.3f}")
```

```
Best Cosine Similarity Score: 0.126
```

[32]:
```
# Enhanced Similarity
response = best_matching_record
print(query,": ", response)
similarity_score = calculate_enhanced_similarity(query, best_matching_record)
print(f"Enhanced Similarity:, {similarity_score:.3f}")
```

```
define a rag store :  A RAG vector store is a database or dataset that contains
vectorized data points.
Enhanced Similarity:, 0.642
```

8.1.3 Augmented input

[33]:
```
augmented_input=query+": "+best_matching_record
```

[34]:
```
print_formatted_response(augmented_input)
```

```
Response:
---------------
```

```
define a rag store: A RAG vector store is a database or dataset that contains
vectorized data points.
---------------
```

8.1.4 Generation

[35]:
```
# Call the function and print the result
llm_response = call_llm_with_full_text(augmented_input)
print_formatted_response(llm_response)
```

```
Response:
---------------
An ARAG vector store, or more generally a vector store, is a specialized type of
database or dataset designed to store and manage vectorized data points. Here's
a more detailed explanation:  1. **Vectorized Data Points**: In the context of
machine learning and data science, data is often transformed into a numerical
format known as vectors. These vectors are essentially arrays of numbers that
represent data in a way that algorithms can process. For example, in natural
language processing, words or sentences can be converted into vectors using
techniques like word embeddings (e.g., Word2Vec, GloVe) or sentence embeddings.
2. **Purpose of a Vector Store**: The primary purpose of a vector store is to
efficiently store, retrieve, and manage these vectorized representations. This
is crucial for tasks that involve similarity search, clustering, or any
operation that requires comparing vectors to find patterns or relationships.  3.
**Applications**: Vector stores are widely used in various applications,
including:    - **Recommendation Systems**: By storing user preferences and item
features as vectors, systems can recommend items that are similar to those a
user has liked in the past.    - **Image and Video Retrieval**: Vectors can
represent visual features, allowing for efficient searching and retrieval of
similar images or videos.    - **Natural Language Processing**: In NLP, vector
stores can be used to find semantically similar texts or to perform tasks like
document clustering.  4. **Key Features**:    - **Scalability**: Vector stores
are designed to handle large volumes of data, making them suitable for big data
applications.    - **Efficiency**: They provide fast retrieval times, which is
essential for real-time applications.    - **Similarity Search**: They often
include algorithms for nearest neighbor search, which is used to find vectors
that are closest to a given query vector.  5. **Examples of Vector Stores**:
There are several tools and platforms that provide vector storage capabilities,
such as FAISS (Facebook AI Similarity Search), Annoy (Approximate Nearest
Neighbors Oh Yeah), and Elasticsearch with its vector search capabilities.  In
summary, an ARAG vector store is a crucial component in modern data-driven
applications, enabling efficient handling and processing of vectorized data for
various analytical and operational purposes.
---------------
```

8.2 3.2.Index-based search

8.2.1 Search Function

[36]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def setup_vectorizer(records):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(records)
    return vectorizer, tfidf_matrix

def find_best_match(query, vectorizer, tfidf_matrix):
    query_tfidf = vectorizer.transform([query])
    similarities = cosine_similarity(query_tfidf, tfidf_matrix)
    best_index = similarities.argmax()  # Get the index of the highest␣
 ↪similarity score
    best_score = similarities[0, best_index]
    return best_score, best_index

vectorizer, tfidf_matrix = setup_vectorizer(db_records)

best_similarity_score, best_index = find_best_match(query, vectorizer,␣
 ↪tfidf_matrix)
best_matching_record = db_records[best_index]

print_formatted_response(best_matching_record)
```

```
Response:
---------------
A RAG vector store is a database or dataset that contains vectorized data
points.
---------------
```

8.2.2 Metrics

[37]:
```
# Cosine Similarity
print(f"Best Cosine Similarity Score: {best_similarity_score:.3f}")
print_formatted_response(best_matching_record)
```

```
Best Cosine Similarity Score: 0.407
Response:
---------------
A RAG vector store is a database or dataset that contains vectorized data
points.
---------------
```

[38]:
```
# Enhanced Similarity
response = best_matching_record
print(query,": ", response)
similarity_score = calculate_enhanced_similarity(query, response)
print(f"Enhanced Similarity:, {similarity_score:.3f}")
```

```
define a rag store :  A RAG vector store is a database or dataset that contains
vectorized data points.
Enhanced Similarity:, 0.642
```

Feature extraction

[39]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

def setup_vectorizer(records):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(records)

    # Convert the TF-IDF matrix to a DataFrame for display purposes
    tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=vectorizer.
 ↪get_feature_names_out())

    # Display the DataFrame
    print(tfidf_df)

    return vectorizer, tfidf_matrix

vectorizer, tfidf_matrix = setup_vectorizer(db_records)
```

	ability	access	accuracy	...	with	within	without
0	0.000000	0.000000	0.000000	...	0.000000	0.260582	0.000000
1	0.000000	0.000000	0.000000	...	0.160278	0.000000	0.000000
2	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000
4	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000
5	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000
6	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000
7	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000
8	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000
9	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000
10	0.186734	0.000000	0.000000	...	0.000000	0.000000	0.000000
11	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000
12	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000
13	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000
14	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000
15	0.000000	0.172624	0.000000	...	0.000000	0.000000	0.000000
16	0.000000	0.317970	0.000000	...	0.258278	0.000000	0.000000
17	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000

```
18  0.000000  0.000000  0.000000  ...  0.000000  0.000000  0.000000
19  0.000000  0.000000  0.000000  ...  0.000000  0.000000  0.000000
20  0.000000  0.000000  0.000000  ...  0.000000  0.000000  0.000000
21  0.000000  0.000000  0.000000  ...  0.192110  0.000000  0.291503
22  0.000000  0.174772  0.000000  ...  0.141963  0.000000  0.000000
23  0.000000  0.000000  0.000000  ...  0.217033  0.000000  0.000000
24  0.000000  0.000000  0.000000  ...  0.134513  0.000000  0.000000
25  0.000000  0.000000  0.228743  ...  0.000000  0.000000  0.000000
26  0.000000  0.000000  0.000000  ...  0.000000  0.000000  0.000000
27  0.000000  0.000000  0.000000  ...  0.000000  0.000000  0.000000

[28 rows x 297 columns]
```

8.2.3 Augmented input

[40]:
```
augmented_input=query+": "+best_matching_record
```

[41]:
```
print_formatted_response(augmented_input)
```

```
Response:
---------------
define a rag store: A RAG vector store is a database or dataset that contains
vectorized data points.
---------------
```

8.2.4 Generation

[42]:
```
# Call the function and print the result
llm_response = call_llm_with_full_text(augmented_input)
print_formatted_response(llm_response)
```

```
Response:
---------------
Certainly! Let's break down the concept of an "ARAG vector store" and what it
means to have a database or dataset containing vectorized data points.  ### ARAG
Vector Store  1. **ARAG**: While "ARAG" isn't a standard term in the context of
vector stores, it might refer to a specific implementation, framework, or
proprietary system related to vector storage. Without additional context, it's
difficult to define "ARAG" precisely, but it could be an acronym or a brand name
associated with a particular vector storage solution.  2. **Vector Store**: A
vector store is a specialized type of database or dataset designed to store and
manage vectorized data points. Vectors are mathematical representations of data,
often used in machine learning and data science to represent features of data
points in a numerical format.  ### Vectorized Data Points  1. **Vectors**: In
the context of data storage, vectors are arrays of numbers that represent data
in a multi-dimensional space. Each dimension corresponds to a feature or
attribute of the data. For example, in natural language processing, words or
sentences can be represented as vectors using techniques like word embeddings
```

```
(e.g., Word2Vec, GloVe) or sentence embeddings.  2. **Vectorization**: This is
the process of converting data into a vector format. For instance, text data can
be transformed into vectors using various algorithms that capture semantic
meaning, allowing for operations like similarity search, clustering, and
classification.  3. **Applications**: Vector stores are crucial in applications
that require efficient similarity search and retrieval. They are used in
recommendation systems, image and video retrieval, natural language processing,
and more. By storing data as vectors, these systems can quickly compute
distances or similarities between data points, enabling fast and accurate
retrieval.  ### Database or Dataset  1. **Database**: In this context, a
database refers to a structured collection of data that is stored and accessed
electronically. A vector database is optimized for storing and querying vector
data, often providing specialized indexing techniques to speed up similarity
searches.  2. **Dataset**: A dataset is a collection of data points that are
typically used for analysis or training machine learning models. When we talk
about a vector dataset, we refer to a collection of data points that have been
transformed into vector format, ready for use in various computational tasks.
In summary, an "ARAG vector store" is likely a system or framework designed to
efficiently store and manage vectorized data points, enabling fast retrieval and
analysis of data based on their vector representations. This is particularly
useful in fields that require handling large volumes of data with complex
relationships, such as machine learning, artificial intelligence, and data
science.
---------------
```

9 4.Modular RAG

Modular RAG can combine methods. For example:

keyword search:Searches through each document to find the one that best matches the keyword(s).

vector search: Searches through each document and calculates similarity.

indexed search: Uses a precomputed index (TF-IDF matrix) to compute cosine similarities.

October 25, 2025 update

`self.documents` is initialized in the fit method to hold the records used for searching and enable the `keyword_search` function to access them without error.

Note on Vector search

In this case, the `def vector_search(self, query):` uses `tfidf_matrix`to increase the vector search performance.

The `def vector_search(self, query):` function could use a brute-force method as implemented in `Section 3.1.Vector search` of this notebook.

[43]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

```
class RetrievalComponent:
    def __init__(self, method='vector'):
        self.method = method
        if self.method == 'vector' or self.method == 'indexed':
            self.vectorizer = TfidfVectorizer()
            self.tfidf_matrix = None

    def fit(self, records):
      self.documents = records  # Initialize self.documents here
      if self.method == 'vector' or self.method == 'indexed':
        self.tfidf_matrix = self.vectorizer.fit_transform(records)

    def retrieve(self, query):
        if self.method == 'keyword':
            return self.keyword_search(query)
        elif self.method == 'vector':
            return self.vector_search(query)
        elif self.method == 'indexed':
            return self.indexed_search(query)

    def keyword_search(self, query):
        best_score = 0
        best_record = None
        query_keywords = set(query.lower().split())
        for index, doc in enumerate(self.documents):
            doc_keywords = set(doc.lower().split())
            common_keywords = query_keywords.intersection(doc_keywords)
            score = len(common_keywords)
            if score > best_score:
                best_score = score
                best_record = self.documents[index]
        return best_record

    def vector_search(self, query):
        query_tfidf = self.vectorizer.transform([query])
        similarities = cosine_similarity(query_tfidf, self.tfidf_matrix)
        best_index = similarities.argmax()
        return db_records[best_index]

    def indexed_search(self, query):
        # Assuming the tfidf_matrix is precomputed and stored
        query_tfidf = self.vectorizer.transform([query])
        similarities = cosine_similarity(query_tfidf, self.tfidf_matrix)
        best_index = similarities.argmax()
        return db_records[best_index]
```

9.0.1 Modular RAG Strategies

[44]:
```
# Usage example
retrieval = RetrievalComponent(method='vector')  # Choose from 'keyword',␣
↪'vector', 'indexed'
retrieval.fit(db_records)
best_matching_record = retrieval.retrieve(query)

print_formatted_response(best_matching_record)
```

```
Response:
---------------
A RAG vector store is a database or dataset that contains vectorized data
points.
---------------
```

9.0.2 Metrics

[45]:
```
# Cosine Similarity
print(f"Best Cosine Similarity Score: {best_similarity_score:.3f}")
print_formatted_response(best_matching_record)
```

```
Best Cosine Similarity Score: 0.407
Response:
---------------
A RAG vector store is a database or dataset that contains vectorized data
points.
---------------
```

[46]:
```
# Enhanced Similarity
response = best_matching_record
print(query,": ", response)
similarity_score = calculate_enhanced_similarity(query, response)
print("Enhanced Similarity:", similarity_score)
```

```
define a rag store :  A RAG vector store is a database or dataset that contains
vectorized data points.
Enhanced Similarity: 0.641582812483307
```

9.0.3 Augmented Input

[47]:
```
augmented_input=query+ " "+ best_matching_record
```

[48]:
```
print_formatted_response(augmented_input)
```

```
Response:
---------------
define a rag store A RAG vector store is a database or dataset that contains
```

```
vectorized data points.
---------------
```

9.0.4 Generation

[49]:
```
# Call the function and print the result
llm_response = call_llm_with_full_text(augmented_input)
print_formatted_response(llm_response)
```

```
Response:
---------------
An "ARAG vector store" refers to a specialized type of database or dataset
designed to store and manage vectorized data points. Let's break down the
concept further:  1. **Vectorized Data Points**: In the context of data science
and machine learning, vectorization is the process of converting data into a
numerical format that can be easily processed by algorithms. Each data point is
represented as a vector, which is essentially an array of numbers. These vectors
can represent various types of data, such as text, images, or any other form of
information that can be numerically encoded.  2. **Purpose of a Vector Store**:
The primary purpose of a vector store is to efficiently store and retrieve these
vectorized data points. This is particularly useful in applications that require
fast similarity searches, such as recommendation systems, natural language
processing tasks, and image recognition. By storing data in a vectorized form,
it becomes easier to perform operations like finding the nearest neighbors or
clustering similar data points.  3. **ARAG Vector Store**: While the term "ARAG"
is not widely recognized in the context of vector stores, it could be a specific
implementation or a proprietary system developed by a particular organization or
for a specific use case. The key aspect of an ARAG vector store would be its
ability to handle and manage vectorized data efficiently.  4. **Applications**:
Vector stores are crucial in various applications, including:    - **Search
Engines**: To improve search results by finding documents or images similar to a
query.    - **Recommendation Systems**: To suggest products or content based on
user preferences and behavior.    - **Natural Language Processing**: To process
and analyze text data by converting words or sentences into vectors.    -
**Computer Vision**: To recognize and categorize images by analyzing their
vector representations.  In summary, an ARAG vector store is a system designed
to store and manage data in a vectorized format, enabling efficient retrieval
and analysis of data points for various applications.
---------------
```

[ ]: